documentation AJL

Table des matières

Introduction

AIDE AUX JEUX DE LETTRES



==== Version 5 ====

AJL (Aide aux Jeux de Lettres) est un programme destiné à tous les amateurs de jeux de lettres : Scrabble(©), résolution ou composition de mots croisés, mot le plus long, textes à contraintes, etc...

Le domaine de AJL est la recherche ultra-rapide de mots qui répondent à toutes sortes de critères de sélection mis à votre disposition dans une interface intuitive, qui ne nécessite pas de connaissance informatique.

AJL exploite ses propres dictionnaires intégrés proposés au téléchargement sur le site JLP Fractware, mais aussi toute liste de mots dont vous disposeriez.

Parcourez cette aide par chapitre, ou en séquence à l'aide des flèches en forme de triangle situées en zone titre, en haut et à droite de chaque page.

- Pour apprendre à vous servir de AJL, voyez le mode d'emploi.
- Vous êtes déjà un habitué d'AJL ? ne manquez pas Le coin des experts.
- Le coin des experts ne vous suffit plus? Alors EEA est fait pour vous.

AJL est un graticiel de JLP Fractware pour Windows 7 et versions ultérieures (testé sur W10).
Si ce logiciel vous est utile, vous pouvez encourager l'auteur par mail.

Documentation à jour pour AJL 5.7 et EEA 5.2

---

remerciements aux contributeurs

remerciements

Remerciements aux contributeurs

Tout d'abord, un grand merci à **Jean-Claude Cocquerez**. C'est lui qui a lancé l'idée initiale de AJL, et lui

aussi qui à la grande époque des images de synthèse (car oui, il fut une époque où les photos numériques n'existaient quasiment pas), a réalisé les images utilisées en écran de démarrage d'AJL.

Merci aussi aux membres de la liste Oulipo pour leurs précieuses idées et leurs encouragements, notamment :

Eric Angelini,
Nicolas Graner,
Alain Zalmanski.

Enfin, un coup de chapeau à certains correspondants et utilisateurs chevronnés d'AJL, partout dans le monde, pour leurs suggestions que j'ai adorées ou détestées (selon la quantité de travail demandée pour les mettre en place) :

Alain Aubin,
Hervé Bourgeois
Pascal Brossard (notamment l'excellente idée de l'appel des définitions des mots en ligne)
Pierre Bressoles
Jacques Dubernard
Jean-Marc Falcoz (stupéfiant expert et testeur de EEA)
Marcel Gravel
Nicole Hannequart (idée de mettre en couleurs les mots selon le dictionnaire de provenance)
Olivier Pourret (courageux débuggeur des 1ères versions 5, idée de la fonction Tiragemot())

L'installateur de AJL a été réalisé avec l'excellent gratuiciel Inno Setup de Jordan Russel.

Cette aide en ligne, tout comme sa version pdf, a été écrite avec HelpNDoc Personal Edition de IBE Software.

AJL est intégralement réalisé en C++ grâce à l'environnement de développement C++ Builder XE2, de la firme Embarcadero pour le code d'interface utilisateur (exécutable AJL.exe), et par l'outillage libre du projet MinGw-w64 pour le code regroupant la logique de calcul (exécutable eea.dll)

historique des révisions

Historique des révisions AJL

Historique des versions 5

5.7.2 (juin 2015) fonction Evalue, extension aux listes des fonctions Min et Max.
5.7.0 (janvier 2015) ajouts mineurs et amélioration de performances. Arrivée du nouveau fichier exécutable **eea.dll** qui doit impérativement être situé dans le même répertoire que le fichier **ajl.exe** de l'application.
5.6.5 (septembre 2014) ajouts mineurs et correctifs.
5.6.4 (août 2014) support de la clause sinonsi (elsif) dans le bloc si - finsi, fonction Anagrammeliste.
5.6.3 (juillet 2014) améliorations du traitement de l'affichage dans les applications multi-tâches
5.6.2 (juin 2014) petites améliorations du compilateur EEA. Correction d'un bug sur Anagramme, ajout de la fonction Joker.
5.6.1 (mai 2014) correction d'un bug affectant le bouton "Poursuivre après le mot"
5.6.0 (mars 2014) extension de EEA à la programmation en listes
5.5.2 (novembre 2013) ajout de la possibilité de modifier la logique des filtres de l'onglet expert.
5.5.1 (octobre 2013) amélioration de performances (gain d'environ 30%) dans EEA.
5.5.0 (juin 2013) introduction de la programmation multi-tâche (multi-threading) dans EEA
5.4.0 (avril 2013) ajout d'un onglet pour le jeu Boogle
5.3.3 (avril 2013) version spéciale Alain Zalmanski.
5.3.2 (mars 2013) possibilité de paramétrer la police de caractères utilisée dans les menus. Compatibilité avec les choix de personnalisation Windows, notamment les thèmes Windows 7 à fond noir.

5.3.1 (février 2013) possibilité de paramétrer la police de caractères utilisée dans les écrans. Amélioration de l'ergonomie du traitement des erreurs de compilation en EEA.
5.3.0 (janvier 2013) livraison de "Nouveau dictionnaire", "Comparaison fichiers", "Extracteur de mots". Extensions du support de dictionnaires en minuscules et accentués. Mise à disposition sur le site d'un dictionnaire en minuscules et accents. Ajouts documentaires. Meilleur support de l'appel au système d'aide. Correctif d'un bug sur les masques. Ajout de la fonction Debutmot()
5.2.1 (décembre 2012) correctif d'un cas de deadlock sur la 5.2.0, correctif sur fonction Tiragemotex
5.2.0 (décembre 2012) optimisation des recherches d'anagrammes pour processeurs multi-cœurs (algorithme fortement parallèle en multi-thread),
5.1.0 (novembre 2012) correctif d'un bug affectant les arrettantque (breakwhile) et arretboucle (breakfor) de boucles imbriquées
5.0.8 (octobre 2012) optimisation de l'usage de la mémoire, ajout fonction Heure()
5.0.7 (juillet 2012) ajout de la fonction Tiragemot()
5.0.6 (juillet 2012) extensions du fonctionnement du filtre joker de l'onglet Expert. Fonction Setjoker ajoutée. Correction de bug sur fonction valeur().
5.0.5 (mai 2012) ajouts et corrections documentaires, correction d'un bug EEA concernant le bloc if - endif
5.0.4 (mai 2012) ajouts et corrections documentaires, correction d'un bug EEA dans certains cas d'utilisation de "Tantque" imbriqués.
5.0.3 (mai 2012) ajouts et corrections documentaires. mise à disposition de l'aide en fichier pdf.
5.0.1 (mai 2012) première version pour Windows 7, 32 et 64 bits. EEA entièrement réécrit.
janvier 2012 à avril 2012 : rachat d'un compilateur Borland C++ Builder récent et portage du source AJL

Plus loin dans le temps, il était une fois...

4.1.5 (novembre 2008) dernière version 4 et abandon provisoire des développements.
4.0.9 (novembre 2003) Le premier AJL en version gratuite. Windows XP.
3.0.5 (février 2000). première version Borland C++ Builder V4. apparition de EEA.
2.6d **(**décembre 1999) dernière évolution en Borland 1.0 .
2.3 (1999) apparition de l'onglet Expert.
2.0 (1998) portage en Borland C++ 1.0 et Windows 32 bits (NT).
1.4 (1998) dernière version pour les Windows 16 bits (Win 95/98). version shareware.
1.0 (1997) première version en Delphi 1.0 pour Windows 3.1. Onglets Inclus et Contient seulement.

Mode d'emploi

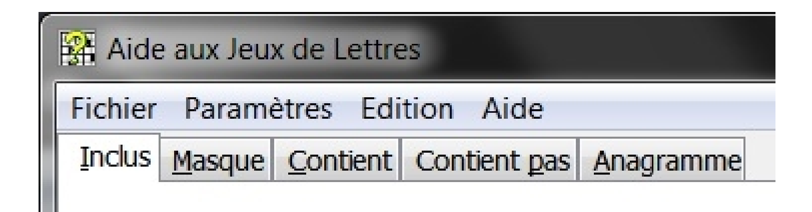
Mode d'emploi de AJL

AJL est essentiellement un outil d'interrogation de dictionnaire, qui permet de sélectionner les mots qui correspondent à votre recherche. De multiples fonctions de recherche sont proposées, chaque type de recherche correspond à un des Onglets de l'interface.

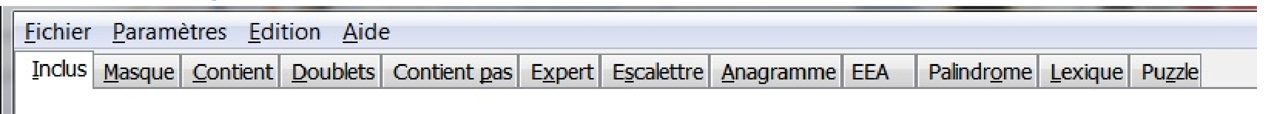
AJL peut lire, comparer ou créer des dictionnaires spécialisés répondant à vos domaines d'intérêt. Le site internet jlpfractware propose gratuitement au téléchargement un ensemble de dictionnaires prêts à l'emploi. Les dictionnaires sont de simples fichiers indépendants, d'extension ".ajl" pour ceux fournis par jlpfractware, ou d'extension ".txt" pour ceux que vous pourriez fabriquer.

L'interface AJL se compose tout d'abord du haut de fiche, qui a deux apparences possibles:

1. **en mode débutant:**



2. **en mode expert:**



Il est fortement recommandé de vous familiariser avec AJL, d'abord en laissant l'interface en mode "débutant".

Voyez ici comment passer du mode "débutant" au mode "expert".

Pour apprendre à vous servir de AJL

découvrez le menu principal (Fichier, Paramètres, Outils, Edition, Aide)

- Voir ici pour une description des fonctions disponibles dans les menus

découvrez les onglets qui correspondent chacun à une fonctionnalité de recherche

- voir ici pour une description des fonctions opérées par les onglets

choisissez le ou les dictionnaires qui vous conviennent

- voir ici une description des 3 types de dictionnaires utilisables dans AJL

Utilisation de l'aide en ligne : Le bouton Aide

- si vous sélectionnez un mot de la liste des mots sélectionnés (un double-clic sur le mot suffit), et appuyez sur Aide, la signification de ce mot est affichée par appel à Internet sur un site web que vous pouvez choisir, voir les Options générales.
- si, dans EEA, c'est une fonction ou une instruction que vous sélectionnez (idem, doucleclic suffit sauf si le mot en question est collé à d'autres mots ou symboles), et appuyez sur Aide, la page d'aide interne AJL correspondante est affichée.
- si aucun texte n'est sélectionné, et que vous appuyez sur Aide, la page d'aide relative à l'onglet actif est affichée.

Vous avez l'impression que le système d'aide ne fonctionne pas ? C'est votre Microsoft Windows qui vous joue des tours. Il faut indiquer à votre Windows que le fichier d'aide est un fichier de confiance, faute de quoi il refusera de l'ouvrir, se contentant d'afficher des rubriques vides. Vous trouverez d'utiles informations à ce sujet dans les pages de ce blog.

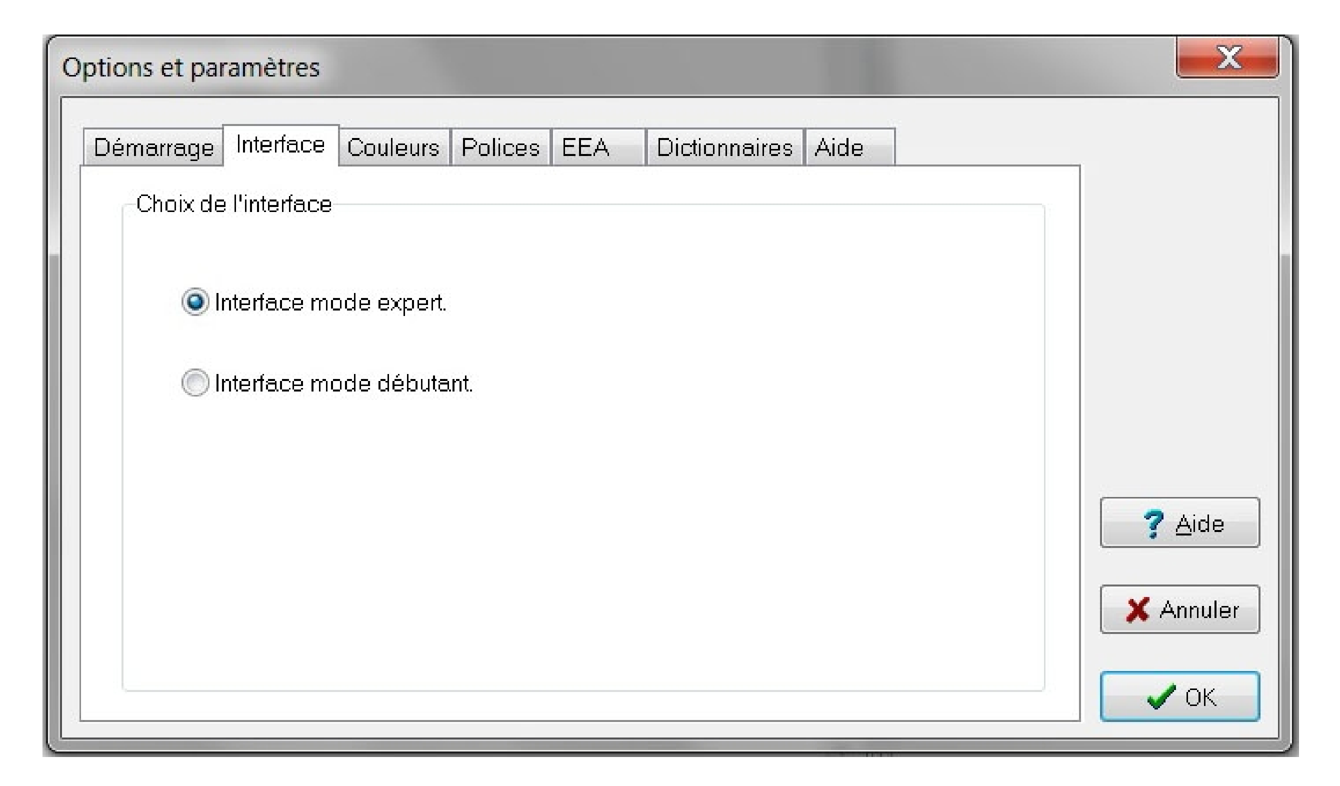
Vous pensez vraiment être déjà un expert AJL?

Alors challengez vous les neurones avec Le coin des experts.

Interface mode expert

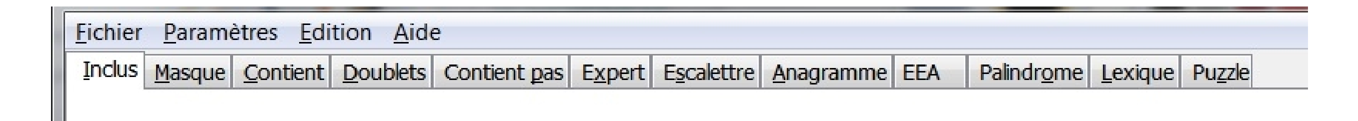
Comment passer du mode "débutant" au mode "expert" ?

Il faut aller dans le menu "**Paramètres**". Choisir "**Options générales**". On obtient une petite fenêtre à onglets qui regroupe tous les réglages possibles dans AJL.
Un de ces onglets se nomme "**Interface**", et comporte un radio-bouton permettant de passer en **mode expert**. Confirmez en appuyant sur OK : les onglets précédemment cachés deviennent visibles !



Onglets

Les onglets AJL



Attention certains onglets ne sont affichés qu'en mode expert. Ils sont repérés par un astérisque (*) dans liste qui suit.
Voyez ici comment passer du mode "débutant" au mode "expert".

AJL dispose de 13 onglets représentant 13 fonctionnalités distinctes.

Inclus

Recherche les mots que l'on peut faire avec un ensemble de lettres, éventuellement complété de jokers

Idéal pour amateurs de ©Scrabble, ©mot le plus long,etc...

Masque
Recherche les mots dont on connaît quelques lettres à certaines positions.
Idéal pour les amateurs de mot-croisés et mots fléchés

Contient
Recherche les mots qui contiennent certaines lettres.

Contient pas
Recherche les mots qui ne contiennent pas certaines lettres.

Anagramme
Recherche les mots ou les phrases qu'il est possible de construire à partir d'un jeu de lettres donné, comme "J'AI RELU DEUX AS DE TEXTE" (Pérec et Queneau ?) anagramme de "AIDE AUX JEUX DE LETTRES".

Expert (*)
Recherche les mots qui satisfont une série de un à six critères largement paramétrables.

Escalettre (*)
Outil de travail sur les listes de mots tels que chaque mot de la liste ne comporte qu'une lettre supplémentaire par rapport au mot précédent. Aucun ordre spécifique n'est exigé pour la suite des lettres de chaque mot.

Doublets (*)
Outil de travail sur les listes de doublets de Carrol, c'est à dire les listes tels que chaque mot de la liste ne diffère que d'une seule lettre par rapport au mot précédent, toutes les autres étant identiques et à la même place dans le mot.

Palindrome (*)
Outil de travail facilitant la constitution de phrases palindromiques, comme "ESOPE RESTE ICI ET SE REPOSE" ou encore "ELU PAR CETTE CRAPULE".

Lexique (*)
Recherche les mots qu'il est possible de construire avec des fragments de mots réutilisables.

Puzzle (*)
Recherche les mots qu'il est possible de construire en assemblant exactement tous les fragments de mots donnés.

Boogle
Recherche les mots qu'il est possible de construire à partir d'une grille ©Boogle.

EEA (*)
Langage informatique intégré, permettant la réalisation de scripts sur les dictionnaires AJL, utilisant une très riche bibliothèque de fonctions travaillant sur les mots. Le chouchou des Oulipiens.

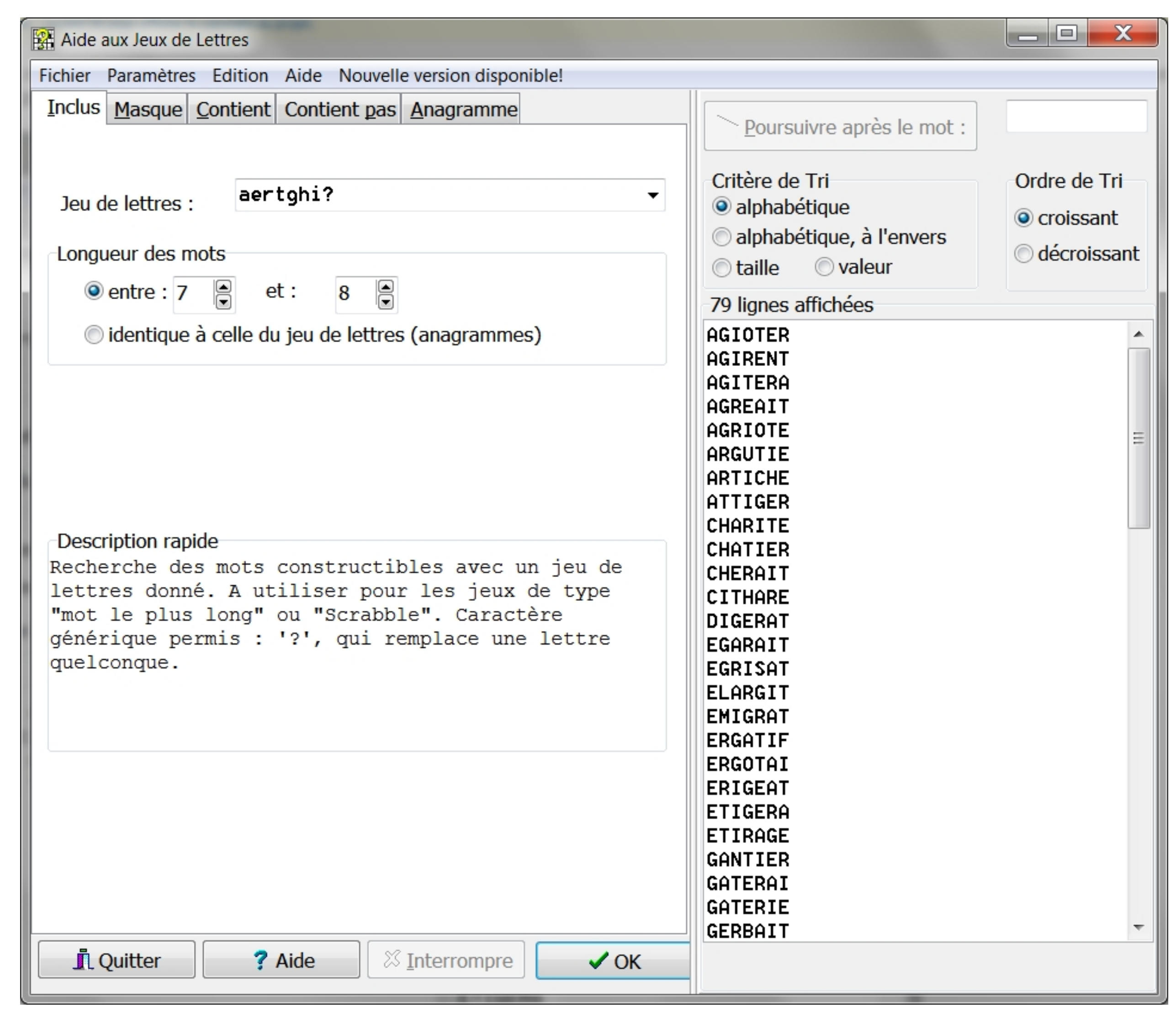
NB: "Scrabble" et "Le mot le plus long" sont des marques déposées par leurs propriétaires respectifs.

Onglet Inclus

Onglet Inclus

Utilisation simple

La fonction Inclus permet de rechercher les mots qu'il est possible de former avec un ensemble de lettres donné, éventuellement complété d'un ou plusieurs jokers "?". Cette fonction de base est idéale pour les jeux "Scrabble" (©) ou "Le mot le plus long" (©), par exemple.



On peut préciser la longueur de mots désirés de deux façons :

- soit chercher les mots contenant toutes les lettres précisées, c'est-à-dire chercher des **anagrammes**, (cocher la case " longueur identique à celle du jeu de lettres")
- soit les mots dont la longueur est comprise entre les valeurs indiquées (cocher la case " longueur entre x et y", et ajuster les valeurs désirées).

Utilisation avancée (mode expert)

Voyez ici comment passer du mode "débutant" au mode "expert".

On peut obliger à puiser les lettres des mots dans l'ordre où elles se présentent dans le jeu de lettres (case à cocher "dans l'ordre gauche->droite", ou dans l'ordre inverse, ou sans tenir compte de l'ordre, ce qui est le choix par défaut). Voir l'exemple.

voir aussi
l'onglet Lexique, qui propose une variation intéressante sur ce thème.

Caractères génériques

Le caractère générique "?" permet de remplacer une lettre quelconque (comme un joker)

Exemples

1. les mots que l'on peut former avec "EAINTRS" en 7 lettres sont: ARISENT, ENTRAIS, INSERAT, RATINES, RESINAT, RIANTES, SATINER, SENTIRA, SERIANT, SERINAT, TANISER, TARSIEN, TRAINES, TRANSIE, TSARINE.
2. "UV?", c'est à dire deux lettres UV et un joker donnent: VAU, VUE et VUS.
3. avec "CREIUTNALB", en mots de 5 lettres respectant l'ordre des lettres gauche->droite du jeu de lettres, on ne peut construire que CRENA, CRETA, RENAL, RITAL.

retour Mode d'emploi

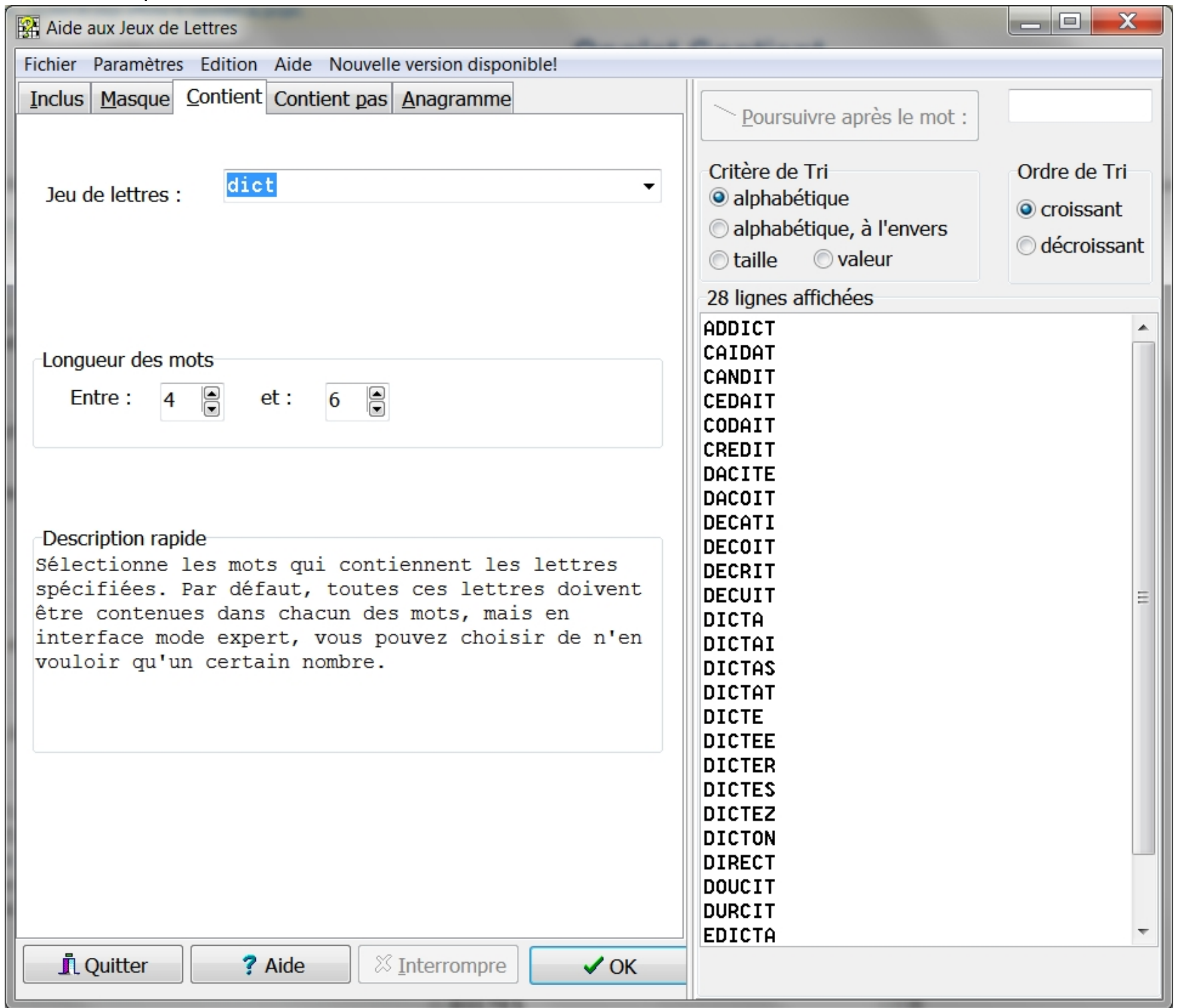
NB: "Scrabble" et "Le mot le plus long" sont des marques déposées par leurs propriétaires respectifs.

Onglet Contient

Onglet Contient

Utilisation simple

La fonction Contient permet de rechercher les mots contenant toutes les lettres spécifiées, et ce dans n'importe quel ordre et à n'importe quelle position. La longueur minimale et maximale des mots recherchés doit être indiquée.



Exemple :

- problème : trouver les mots les plus courts contenant toutes les voyelles, comme AUTONYMIE
- solution : spécifiez les 6 voyelles en jeu de lettres, triez par taille croissante

Utilisation avancée (mode expert)

Voyez ici comment passer du mode "débutant" au mode "expert".

En jouant sur le compteur prévu à cet effet, on peut élargir la recherche et préciser que l'on souhaite sélectionner les mots qui contiennent au moins un certain nombre des lettres parmi toutes celles spécifiées. (Par défaut, ce nombre est ajusté au nombre de lettres du jeu de lettres, de sorte que **toutes** les lettres du jeu de lettres doivent être incluses dans les mots).

Exemple :

- problème : trouver les mots de 6 lettres contenant 5 des 6 voyelles comme OISEAU, YOUPIE
- solution : tapez "AEIOUY" en jeu de lettres, et utilisez le compteur à 5 pour préciser que l'on accepte les mots n'ayant que 5 lettres parmi les 6 voyelles. N'oubliez pas de régler la taille des mots à 6 lettres.

On peut aussi obliger à ce que les lettres du jeu de lettres se retrouvent dans le même ordre dans les mots qui les contiennent. Il suffit de cocher la case "Lettres des mots dans le même ordre que dans le jeu de lettres" correspondant à ce choix.

Caractères génériques

Les caractères génériques ne sont pas supportés par cette fonction

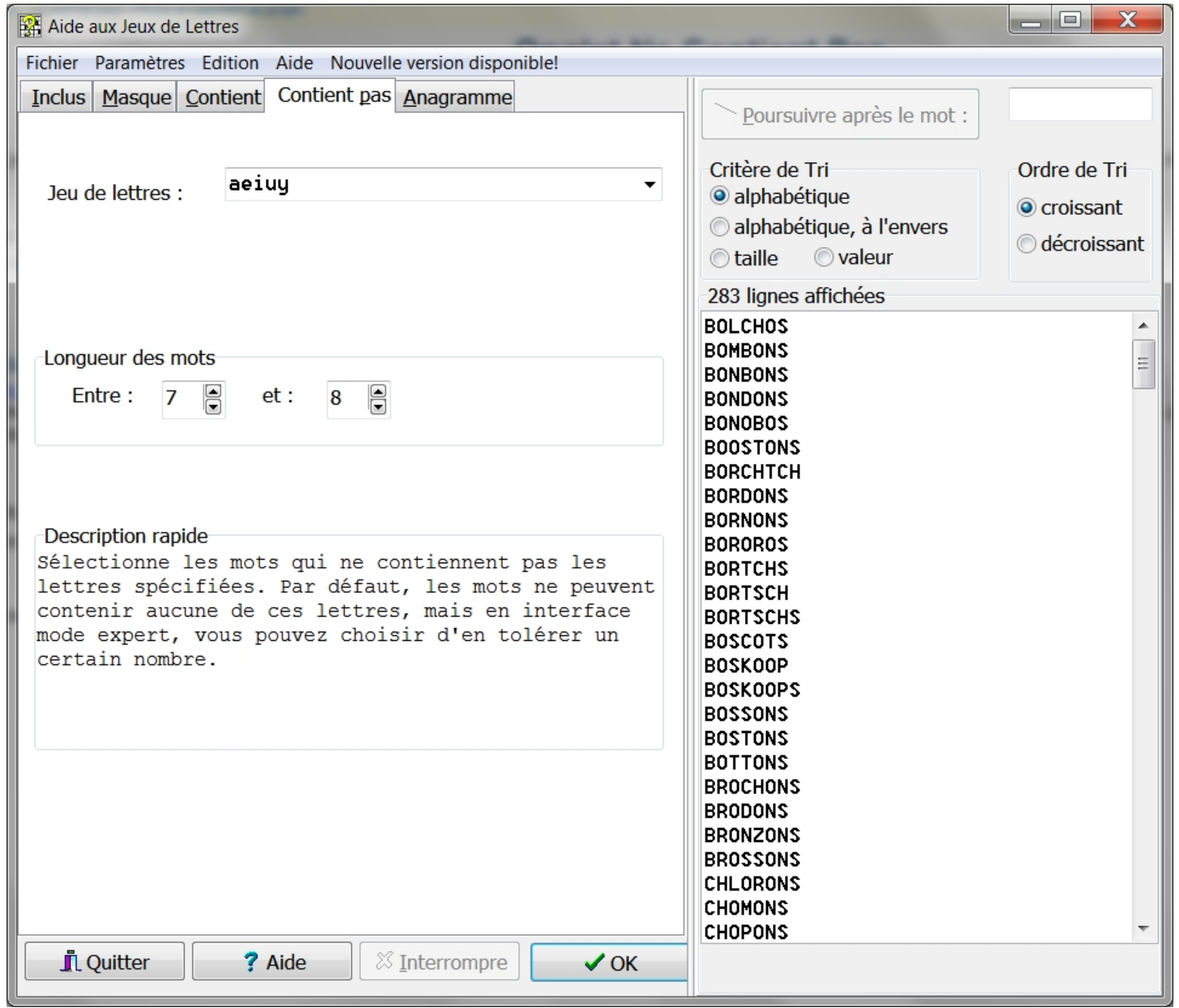
retour Mode d'emploi

Onglet Ne Contient Pas

Onglet Ne Contient Pas

Utilisation simple

La fonction Ne Contient Pas permet de rechercher les mots ne contenant aucune des lettres spécifiées dans le jeu de lettres. La longueur minimale et maximale des mots recherchés doit être indiquée.



Exemple :

- problème : trouver les plus longs mots ne contenant aucune autre voyelle que E, comme DEREGLEMENTERENT
- solution : spécifiez "AIOUY" comme jeu de lettres, et triez par taille décroissante.

Utilisation avancée (mode expert)

Voyez ici comment passer du mode "débutant" au mode "expert".
En jouant sur le compteur prévu à cet effet, on peut élargir la recherche et préciser que l'on veut sélectionner les mots qui contiennent au plus un certain nombre des lettres parmi celles spécifiées dans le jeu de lettres. (Par défaut, ce nombre est zéro, de sorte que les mots cherchés ne peuvent contenir **aucune** des lettres du jeu de lettres.).

Exemple :

- problème : trouver les mots de 6 lettres ne contenant pas plus d'une consonne, comme OISEAU, AIEULE, INOUIE, YOUYOU etc...

- solution : mettez la liste des consonnes "BCDFGHJKLMNPQRSTVWXZ" dans le jeu de lettres, et indiquez 1 en nombre de lettres tolérées.

Caractères génériques
Les caractères génériques sont interdits

retour Mode d'emploi

Onglet Masque

Onglet Masque

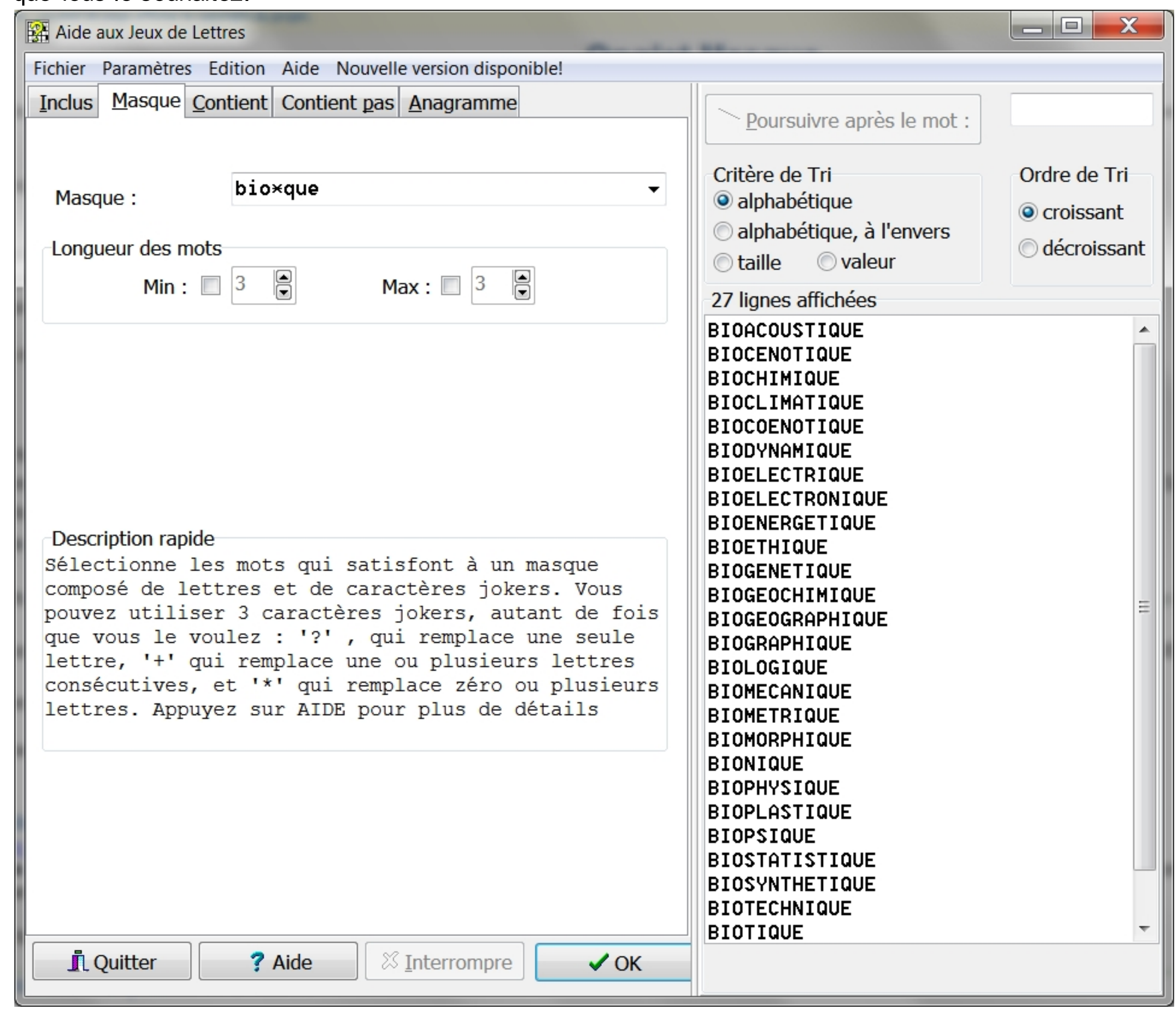
La fonction Masque permet de rechercher les mots dont certaines lettres seulement sont connues, à certaines positions. Cette fonction est idéale pour les cruciverbistes ou amateurs de mots fléchés.

Utilisation simple

Il suffit de déterminer un masque auquel les mots cherchés devront se conformer. Un masque est constitué :

1. de lettres,
2. de caractères joker "?" qui représentent une (une seule) lettre quelconque,
3. de caractères joker "+" qui représentent une suite de plusieurs lettres du mot (au moins une lettre),
4. de caractères joker "*" qui représente aussi une suite de lettres du mot, éventuellement vide.

Vous pouvez utiliser dans un même masque autant de caractères jokers des 3 types, et autant de lettres que vous le souhaitez.



Exemples :

- Le masque "**O?S**" donne tous les mots de 3 lettres commençant par un O et finissant par un S.
- Le masque "**S?R?T?**" donne tous les mots de six lettres exactement commençant par un S, ayant un R en troisième position, un T en cinquième :SCRUTE, SIROTA, SURETE, etc...
- le masque "**O*S**" donne tous les mots commençant par un O et finissant par un S. "OS" en fait aussi

partie, car * peut être vide.

- "0+S" donne la même liste de mots, sauf "OS", car + doit masquer au moins une lettre.

- "?O*QUE" donne les mots commençant par une lettre quelconque, ayant un O en seconde position, et se terminant par QUE. Comme BOTANIQUE, ou LORSQUE. Il peut y avoir n'importe quel nombre de lettres entre le O et le QUE, et même zéro lettres comme dans LOQUE.
- "?O??QUE" donne les mots commençant par une lettre quelconque, ayant un O en seconde position, se poursuivant par deux lettres, et se terminant par QUE. Comme LORSQUE. Il ne peut y avoir que 2 lettres entre le O et le QUE
- "*S*R*T*" donne tous les mots comportant n'importe où, mais dans cet ordre, un S, puis un R, puis un T. Comme DESERT, ou SECRETAIRE

Il est aussi possible de préciser la longueur des mots recherchés, grâce aux paramètres prévus à cet effet.

Utilisation avancée (mode expert)

Voyez ici comment passer du mode "débutant" au mode "expert".
il est possible de **nommer les caractères jokers "?"** en les suffixant par un chiffre entre 1 et 26. Cette possibilité n'est utile que dans le cas où vous voulez imposer que la lettre représentée par le joker se trouve en plusieurs endroits du mot. Dans ce cas précis, vous utiliserez plusieurs fois le même joker nommé dans un même masque.

Choisissez alors le mode de fonctionnement que vous souhaitez, en choisissant un des trois radio-boutons de la boîte

"Règles concernant les caractères jokers ?1 à ?26".

1. **Pas de règle particulière** : Il n'y a aucune contrainte d'utilisation de ces jokers. Dans ce mode, un joker nommé "?1" par exemple a exactement le même fonctionnement qu'un joker simple "?", tel que décrit dans le fonctionnement simple. Ce choix n'est utile que pour la fonction MASQUE de l'onglet Expert. Voyez l'onglet Expert.

2. **Un joker répété doit toujours masquer la même lettre.**

Exemples :

- problème : trouver les mots de 4 lettres de la forme "?LL?" (une lettre, deux L, une lettre), où première et dernière lettre sont identiques, comme ALLA et ELLE.
- solution : "?1LL?1" . Remarquez bien qu'un masque simple "?LL?" aurait aussi trouvé ALLO.
- problème : trouver les mots qui commencent et finissent par les mêmes deux lettres : ALLUVIAL, ONCTION, SEMEUSE, etc...
- solution : le masque "?1?2*?1?2"

3. **A un joker correspond une seule lettre, et réciproquement** :

Comme la règle précédente, mais en plus une même lettre ne peut pas se retrouver dans deux jokers différents.

Exemples :

- problème : trouver les mots de 4 lettres de la forme "E??E" où seconde et troisième lettres sont différentes, comme ELFE, ETRE, etc...
- solution : "E?1?2E" . Notez que dans ce 3ème mode on rejette ELLE car le 'L' ne peut être

à la fois dans ?1 et dans ?2. Notez aussi que dans le 2ème mode précédent, ce mot ELLE aurait été accepté.

Voyez aussi le coin des experts

retour Mode d'emploi

Onglet Expert

Onglet Expert

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert".

Cet onglet permet de combiner plusieurs requêtes en une seule exploration du dictionnaire.

Par défaut, Il fonctionne simplement de haut en bas : la sortie du premier filtre est transmise au filtre suivant, et ainsi de suite. Les mots qui passent tous les filtres sont finalement affichés. Il est possible de modifier ce fonctionnement et de combiner différemment les résultats des filtres.

Cela vous donne la possibilité de réaliser des requêtes extrêmement sophistiquées! Vous trouverez au coin des experts une série d'exemples dont vous pourrez utilement vous inspirer.

L'onglet Expert propose 6 filtres successifs, avec leurs six paramètres associés. Chaque filtre peut être choisi, par une boite déroulante, parmi la liste suivante:



En outre, les deux zones de saisie des longueurs mini et maxi des mots disposent chacune d'une case à cocher, activée par défaut. Si vous désactivez la case, vous demandez à AJL de ne pas prendre en considération la valeur min ou max spécifiée

Description des 16 filtres disponibles

- ### (Aucun)

Comme son nom l'indique, il ne filtre rien, c'est à dire qu'il laisse tout passer. C'est le filtre par défaut.

- ### Inclus

Ce filtre correspond à l'onglet INCLUS du même nom. Il sélectionne les mots constructibles avec un jeu de lettres, que vous spécifiez tout simplement dans la zone de saisie "Paramètres", juste à droite du filtre.

- ### Contient

Ce filtre correspond à l'onglet CONTIENT du même nom. Il sélectionne les mots qui contiennent tout ou partie des lettres d'un jeu de lettres, que vous spécifiez simplement dans la zone de saisie "Paramètres", juste à droite du filtre.

Pour demander qu'un nombre minimum de lettres doit être contenu dans les mots, indiquez ce nombre après le jeu de lettres, séparé de celui-ci par une virgule.
Exemple :

JKLM,3

Demande que 3 des 4 lettres JKLM soient contenues dans les mots.

Si vous omettez ce paramètre :
Exemple :

XYZ

celà signifie que vous voulez que **toutes** les lettres soient contenues dans les mots sélectionnés.

- ## Contient pas

Ce filtre correspond à l'onglet NE CONTIENT PAS du même nom. Il sélectionne les mots qui ne contiennent aucune ou peu des lettres d'un jeu de lettres, que vous spécifiez simplement dans la zone de saisie "Paramètres", juste à droite du filtre.
Pour demander qu'un nombre maximum de lettres soit contenu dans les mots, indiquez ce nombre après le jeu de lettres, séparé de celui-ci par une virgule.
Exemple :

JKLM,1

Demande que 1 au plus des 4 lettres JKLM peut être contenue dans les mots.

Si vous omettez ce paramètre
Exemple :

XYZ

celà signifie **qu'aucune** de ces lettres ne doit être contenue dans les mots sélectionnés.

- ## Lexique

ce filtre correspond à l'onglet Lexique. Il suffit d'indiquer dans la zone de saisie la liste des fragments de mots à utiliser comme lexique.
Exemple

a,bi,en,l

pour trouver les mots composables en utilisant, autant de fois que nécessaire, tout ou partie de ces fragments (ALBIEN, LABIAL, ALLA, etc...)

- ## Masque

Ce filtre correspond à l'onglet MASQUE du même nom. Il sélectionne les mots qui correspondent à un masque constitué de lettres et des caractères jokers "**?**" (1 lettre), "**+**" (1 ou plusieurs lettres) et "*****" (zéro ou plusieurs lettres), que vous spécifiez simplement dans la zone de saisie "Paramètres", juste à droite du filtre.

Les particularités des jokers nommés ?1 à ?26 de l'onglet masque sont ici aussi utilisables, et le choix de paramètre intitulé dans cet onglet "Règles concernant les caractères jokers ?1 à ?26" est respecté.
Attention, si vous utilisez plusieurs filtres Masque à la suite, et que ces filtres partagent un de ces jokers, les règles concernant les caractères jokers répétés s'appliquent aussi !

Pour une utilisation conjointe avec le filtre MOT, tous les caractères génériques sont différentiables en les suffixant par un chiffre : *1,*2, ...,*26, ou +1,... +26, ou ?1, ?2, ...,?26. Lorsqu'un mot franchit avec succès un filtre, ces symboles contiennent les lettres qu'elles ont "masqué".
Voir pour plus de détails le coin des experts.

- ## Sauf masque

Ce filtre est l'opposé du filtre masque. Il sélectionne donc les mots qui ne correspondent pas au masque spécifié.

- ## Valeur min

La boîte de dialogue **Paramètres->Valeur des lettres** permet de donner à chaque lettre une valeur numérique. Ce filtre vous permet de sélectionner les mots dont la valeur totale des lettres est supérieure ou

égale à un minimum, que vous spécifiez dans la zone paramètres, à droite du filtre.

- ## Valeur max

La boîte de dialogue **Paramètres->Valeur des lettres** permet de donner à chaque lettre une valeur numérique. Ce filtre vous permet de sélectionner les mots dont la valeur totale des lettres est inférieure ou égale à un maximum, que vous spécifiez dans la zone paramètres, à droite du filtre.

- ## Mot

Le filtre MOT fonctionne en collaboration avec le filtre MASQUE. Dans le filtre MASQUE, vous pouvez différencier les caractères génériques utilisés en les suffixant par un chiffre (*1,...,*26; +1,. . . +26 ; 1,...,?26). Quand un mot franchit un tel filtre MASQUE, AJL mémorise les valeurs qu'ont prises chacun des caractères génériques utilisés.

Le filtre MOT sert à reconstituer un autre mot avec ces valeurs, et à vérifier qu'il existe au dictionnaire. Si le test est positif (le mot est bien dans le dictionnaire), le mot en cours de traitement est sélectionné, sinon, il est rejeté. Voyez bien que c'est le mot du dictionnaire en cours de traitement qui est rejeté ou sélectionné, et non le mot reconstitué dans le filtre MOT.

Exemple
Soit le filtre MASQUE `*1LT*2` qui sélectionne entre autres le mot SOULTE. Pour ce mot, AJL valorise les deux variables : *1 = SOU et *2 = E. Si votre filtre MOT est `*1FR*2`, le mot SOULTE passe également avec succès ce 2ème filtre car le mot reconstitué par le filtre MOT : "*1"+"FR"+"*2" = "SOU"+"FR"+"E" = "SOUFRE" est un mot du dictionnaire.
Voir pour plus de détails le coin des experts.

- ## Sauf mot

Ce filtre est l'opposé du filtre mot. Il reconstitue un mot selon le même mécanisme que le filtre mot, et vérifie sa présence au dictionnaire. A l'inverse du filtre mot, le test et considéré comme positif, et le mot est sélectionné, si le mot reconstitué n'est pas présent dans le dictionnaire.

- ## Joker

JOKER n'est pas un véritable filtre. Il sert à restreindre les lettres possibles des caractères joker "?" ou"+" ou "*" utilisés dans un filtre MASQUE situé après lui dans la liste des 6 filtres possibles. Ainsi, pour que le caractère joker "?" ne puisse être remplacé que par une des cinq lettres "HIJKL", il suffit de préciser cette liste de lettres en paramètre du "filtre" JOKER.

Si votre filtre MASQUE utilise plusieurs caractères jokers différenciés (?1, ?2, +1, +2..., *1, *2, etc...), il vous suffit de suffixer la liste de lettres par une virgule suivie du numéro du joker qui vous intéresse : par exemple "ABCDEF,1" pour imposer que les jokers ?1 +1 et *1 ne puissent représenter que ces 6 lettres.

- ## Avant

Ce filtre ne laisse passer que les mots placés strictement avant la valeur spécifiée.

- ## Après

Ce filtre est l'opposé du filtre Avant. Il ne laisse passer que les mots placés après la valeur spécifiée, ainsi que le mot éventuellement égal à cette valeur.

Modification de la logique des filtres

Par défaut, les mots sélectionnés et affichés doivent passer avec succès chacun des 6 filtres. C'est cette condition qui est exprimée, par défaut, dans le champ "logique des filtres" : **1 et 2 et 3 et 4 et 5 et 6** qui signifie qu'un mot doit être sélectionné par le filtre **1 et** aussi par le filtre **2 et** aussi par le filtre **3 et** aussi par le filtre **4 et** aussi par le filtre **5 et** aussi par le filtre **6**.

Vous pouvez librement exprimer toute autre forme de condition, en utilisant les mots clés "**et**" et "**ou**" et si besoin des parenthèses. Exemples :

- **1 ou 2**

pour exprimer que les mots affichés doivent être ceux sélectionnés soit par le filtre 1, **ou** soit par le filtre 2.

- **(1 et 2) ou 3**

pour exprimer que les mots affichés doivent être ceux sélectionnés soit par le filtre 1 **et** aussi par le filtre 2, **ou** soit par le seul filtre 3.

- **(1 ou 2) et (3 ou 4)**

pour exprimer que les mots affichés doivent être ceux sélectionnés premièrement par le filtre 1 **ou** par le filtre 2, **et** deuxièmement aussi par le filtre 3 **ou** par le filtre 4

Le bouton "**réinit filtres**" permet de remettre la valeur du filtre à la valeur par défaut **1 et 2 et 3 et 4 et 5 et 6**

Voir aussi le coin des experts.

retour Mode d'emploi

Le coin des experts

Le coin des experts

Le coin des experts expose de nombreux exemples de recherches complexes que l'Onglet Expert peut mener à bien.

Un grande partie des astuces exposées repose sur les "variables", c'est à dire les fragments de mots d'une lettre nommés ?1, ?2, ?3, etc..., ?26 et les fragments de plusieurs lettres *1, etc..., *26 ou +1, +2, ... +26. Ces variables n'ont aucune valeur prédéfinie à l'entrée du premier filtre, et s'utilisent en trois temps :

1) OPTIONNELLEMENT, vous pouvez limiter les caractères permis pour les variables d'une lettres ?1, ?2...?26 et par les variables de plusieurs lettres *1...*26 ou +1...+26. Par défaut toutes les lettres sont permises. Pour limiter les lettres permises, utilisez le pseudo-filtre JOKER. Par exemple "AEIOUY,1" limite aux 6 voyelles les caractères joker ?1, +1, *1 .

2) Les variables sont valorisées au passage par le filtre MASQUE qui les emploie : Un filtre MASQUE les utilise pour définir un modèle des mots, où les variables de type ? constituent un 'joker" valable pour une unique lettre, et une variable de type * n'importe quel nombre de lettres, y compris zéro lettres. Quand un mot passe avec succès le filtre, c'est que chaque joker peut être identifié à un fragment du mot. Ceci se concrétise par la valorisation de chaque variable utilisée dans le filtre au fragment correspondant.

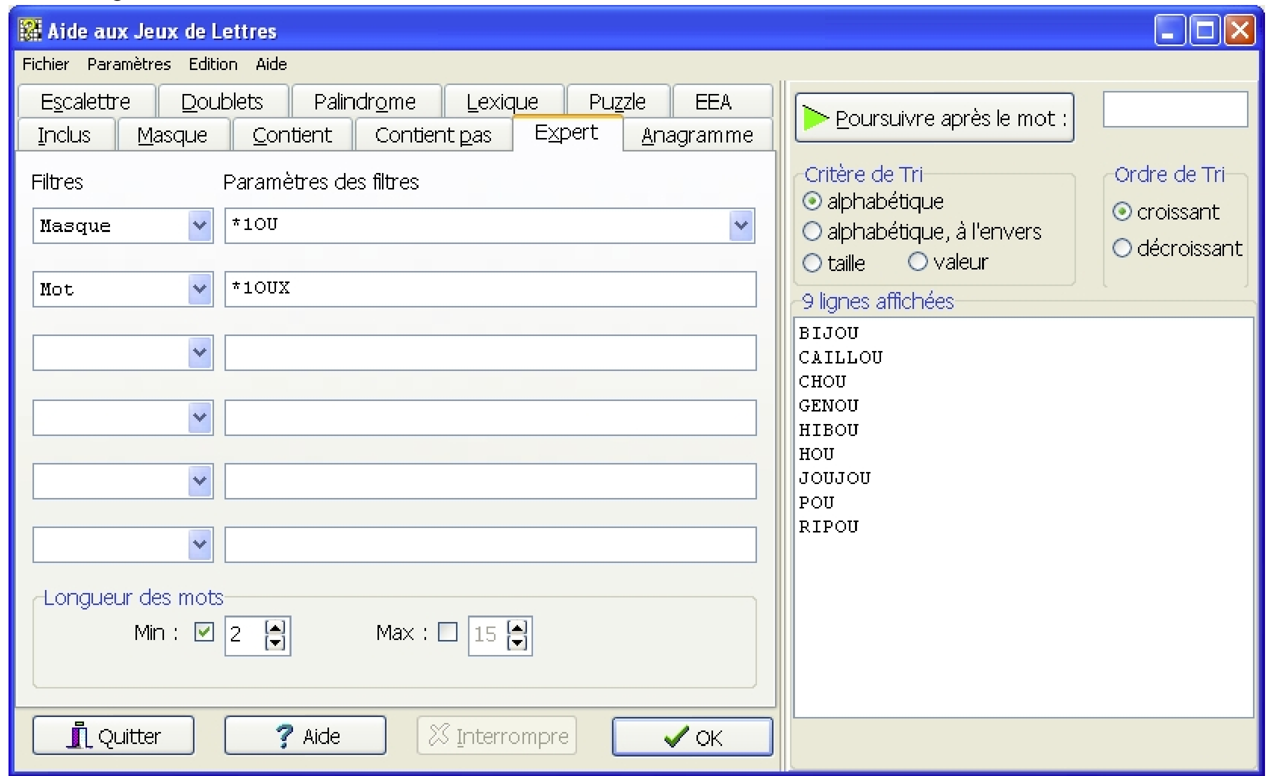
Exemple : "B?1T*1E" utilise deux jokers ?1 et *1. Le mot BATISSE correspond à ce masque, et les variables sont valorisées de la sorte : **?1 = "A"** et ***1 = "ISS"**. On a bien "B?1T*1E" = "B"+"?1"+"T"+"*1"+"E" = B+A+T+ISS+E = BATISSE.

3) elles peuvent être réutilisées dans un filtre MOT qui sert à reconstruire une chaîne de caractères avec des variables et des lettres, et à vérifier si ce mot est présent au dictionnaire. Dans la suite de l'exemple ci dessus, le filtre mot "?1B?1*1E" construit la chaîne de caractères "?1"+"B"+"?1"+"*1"+"E" = A+B+A+ISS+E = ABAISSE qui est accepté, car présent au dictionnaire. Notez bien que si vous avez d'autres filtres suivant celui ci, c'est toujours le mot d'origine BATISSE qui leur est présenté, et non pas ABAISSE.

Exemples

Recherche des mots dont le pluriel est en "OUX"

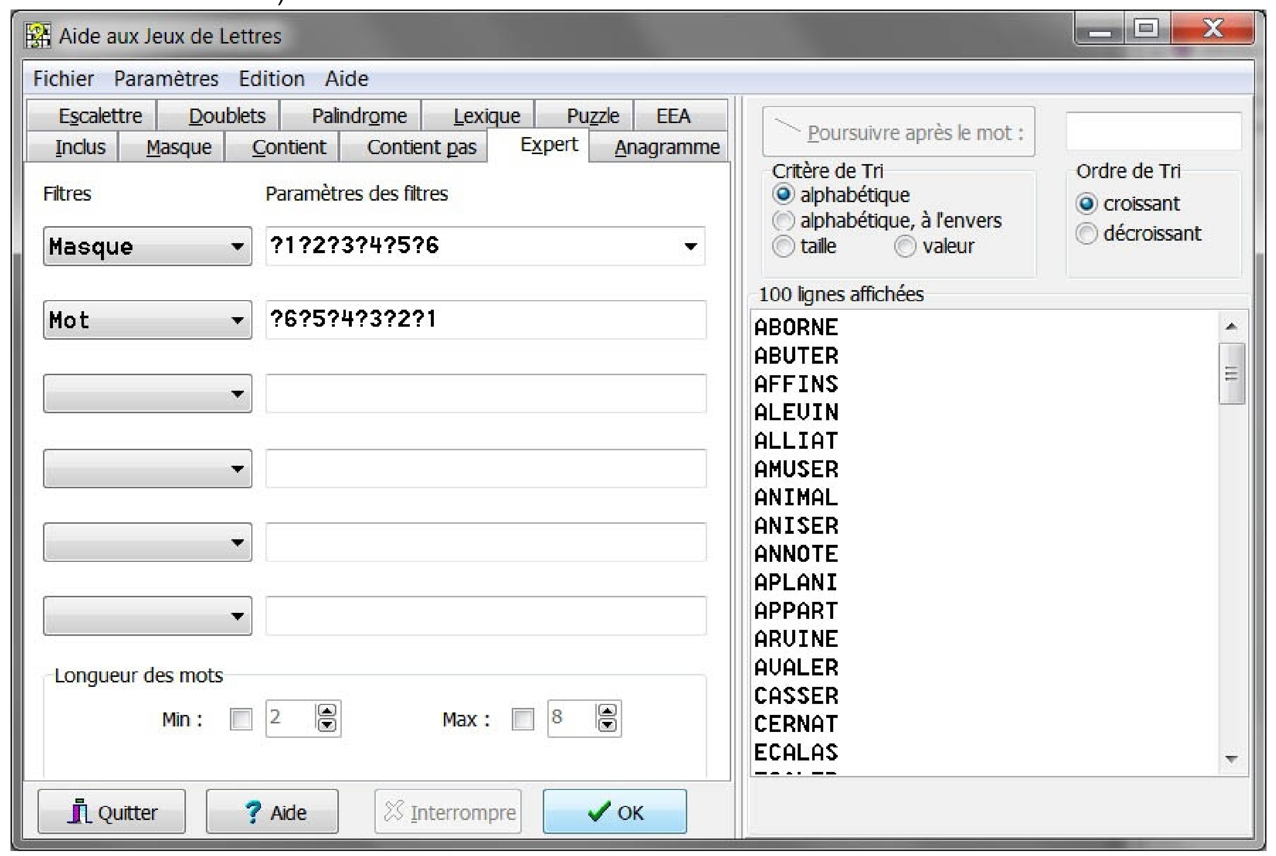
Vous désirez rechercher les mots en OU dont la forme plurielle s'écrit avec un X. Les célèbres hiboux, cailloux, genoux... Deux filtres suffisent:



Le premier filtre les mots qui se terminent en "OU" et valorise au passage la variable "*1" au groupe de lettres précédant le "OU". Le second vérifie la présence au dictionnaire du mot formé par le contenu de la variable *1 complété de "OUX"

Recherche des mots anacycliques

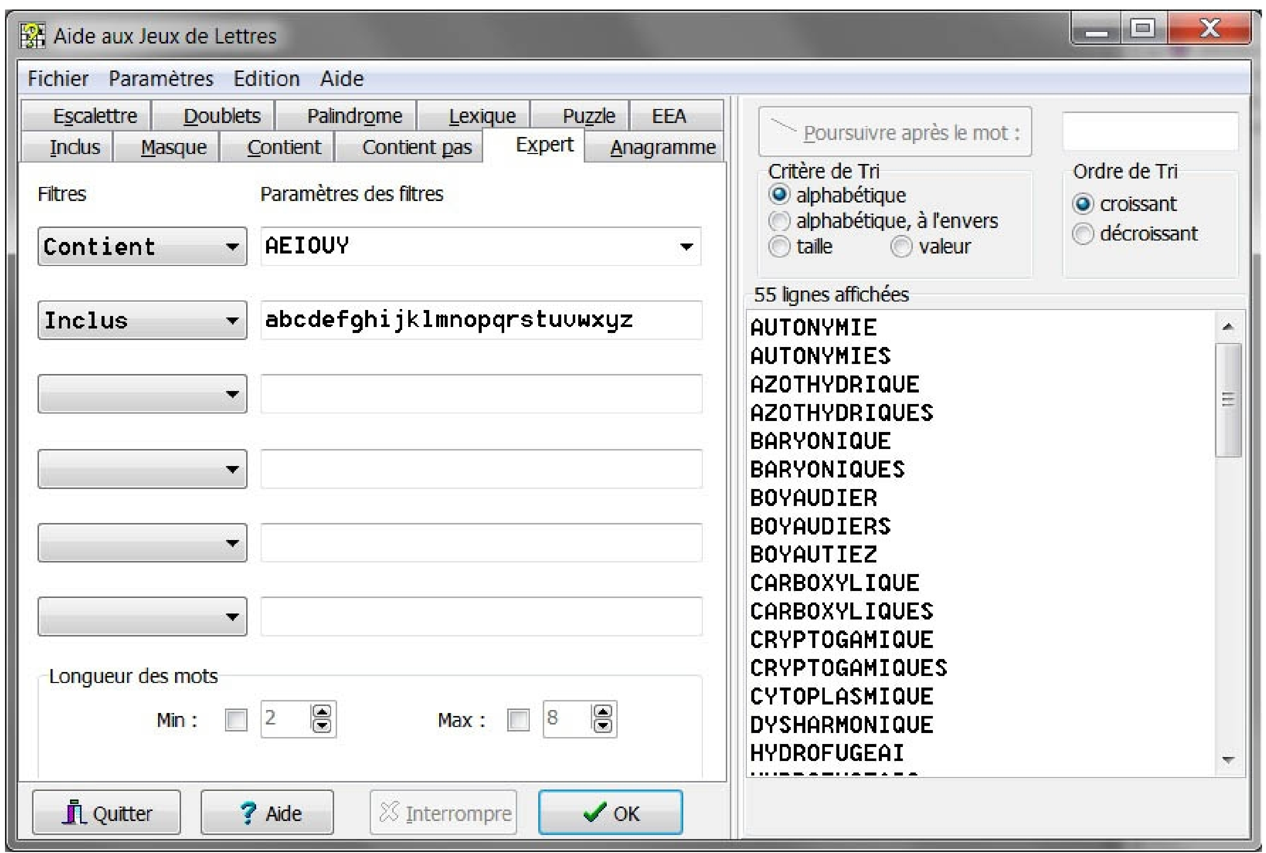
Vous désirez rechercher les mots de 6 lettres anacycliques (qui se lisent dans les deux sens, comme ABORNE<-> ENROBA). Deux filtres suffisent:



1) Le filtre MASQUE ne filtre rien, mais chaque mot qui passe valorise la variable ?1 à la 1ère lettre, ?2 à la seconde, etc.
2) Le filtre MOT vérifie que le mot à l'envers est un mot valide.

Recherche des mots contenant toutes les voyelles sans répétition d'aucune lettre

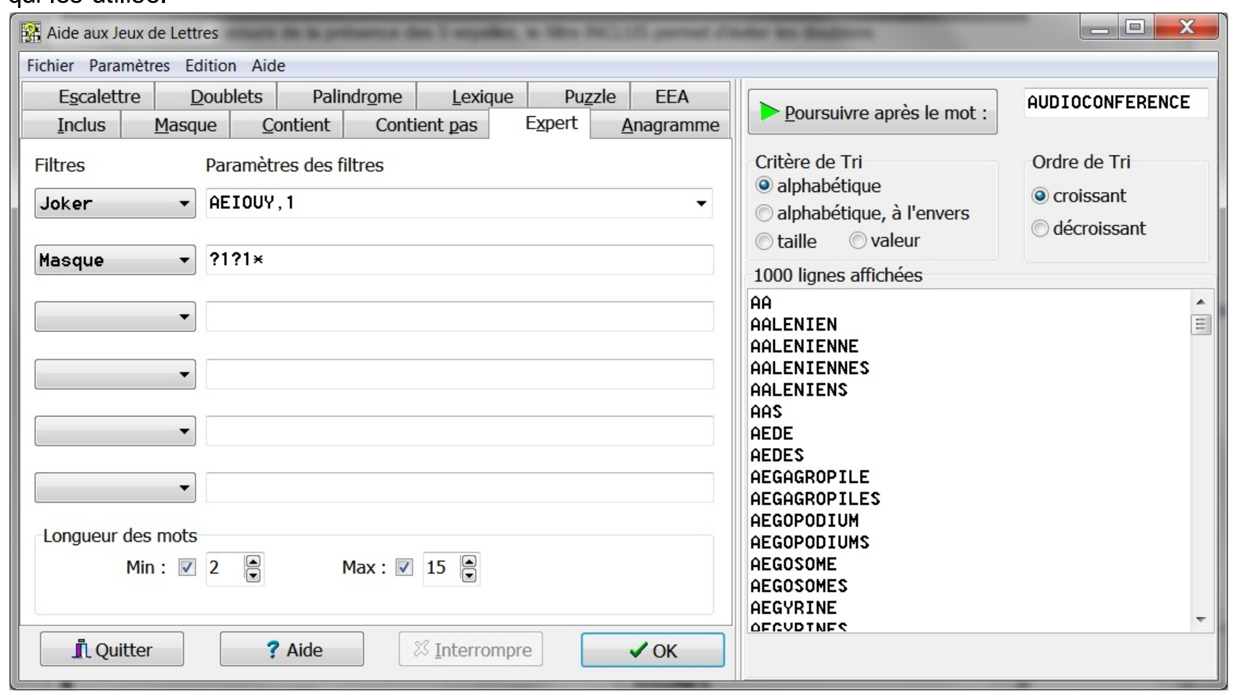
Vous désirez rechercher les mots contenant aeiouy, et n'ayant aucune lettre en double . Deux filtres suffisent:



le filtre CONTIENT s'assure de la présence des voyelles, le filtre INCLUS permet d'éviter les doublons

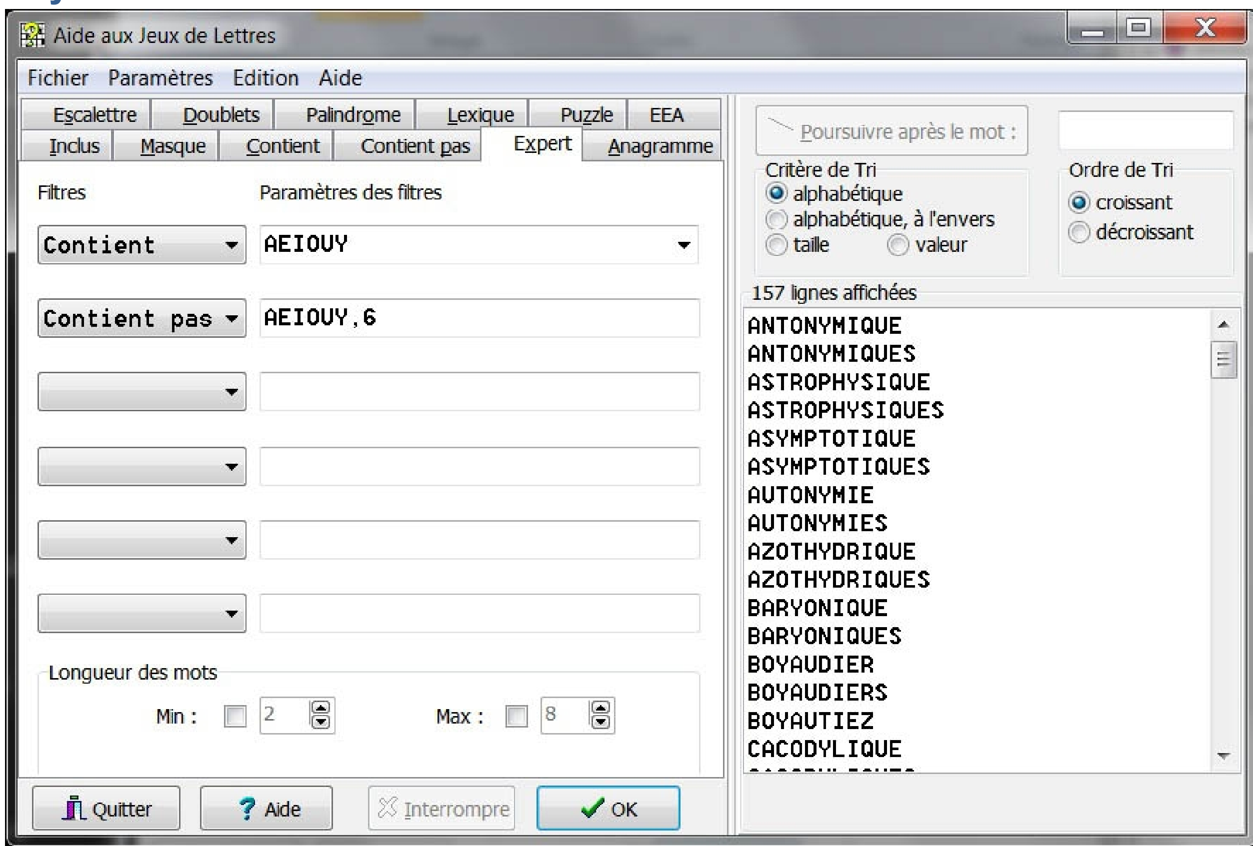
Recherche des mots dont les deux premières lettres sont des voyelles

On utilise un caractère variable "?1", dont on réduit le champ d'action par un filtre JOKER, et un MASQUE qui les utilise.



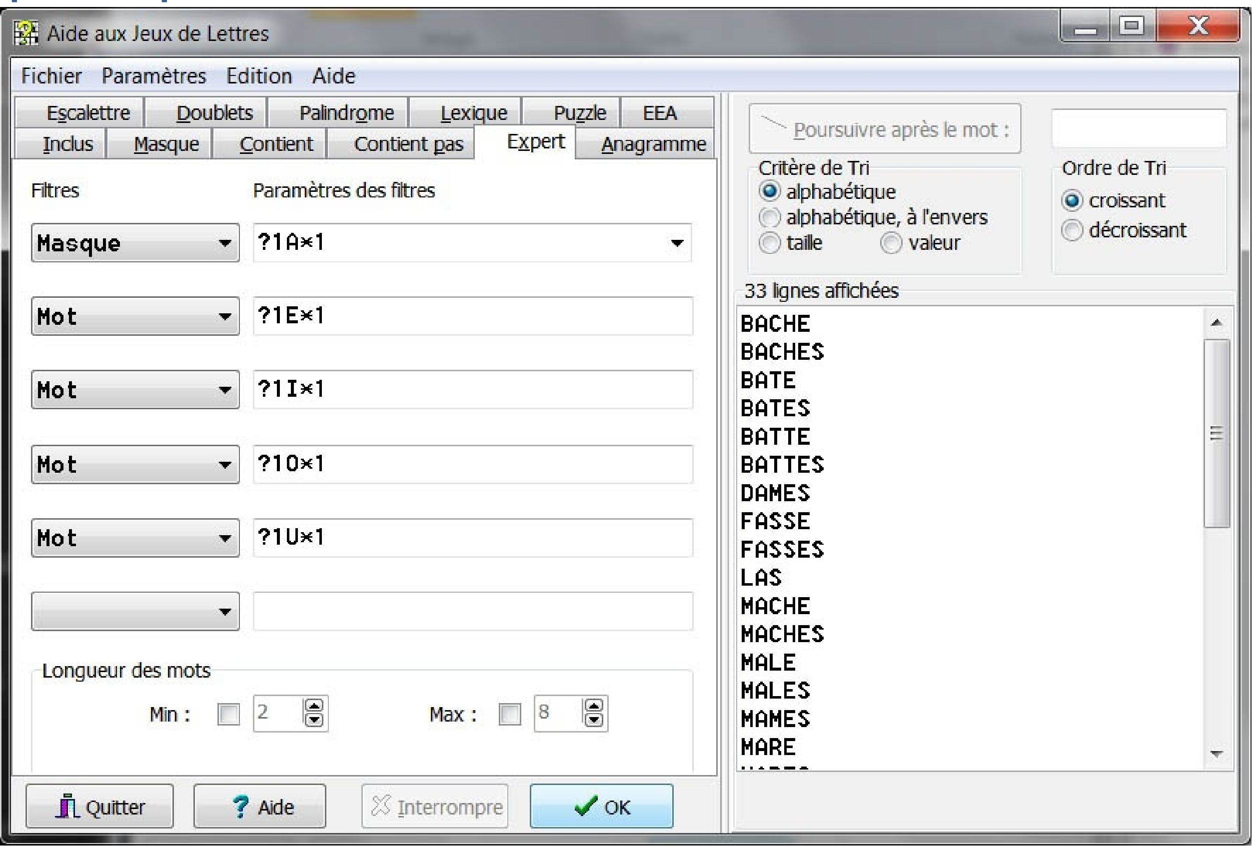
Le joker ?1 est limité aux lettres AEIOUY grâce au "filtre" JOKER. Remarque : le masque utilise deux fois ?1, et comme on ne désire pas imposer que les deux premières lettres soient identiques, on choisira "Pas de règle particulière" dans le paramétrage du mode de fonctionnement des jokers, dans l'onglet Masque.

Recherche des mots contenant exactement une fois chacune des voyelles



Le premier filtre s'assure de la présence des 6 voyelles. Reste à éliminer les doublons, rôle du second filtre, qui n'autorise pas plus de 6 occurences des lettres aeiouy.

Recherche des mots dont la voyelle en 2ème lettre peut être quelconque



On trouve BACHE (BECHE, BICHE, BOCHE, BUCHE), FASSE (FESSE, FISSE, FOSSE, FUSSE) etc...

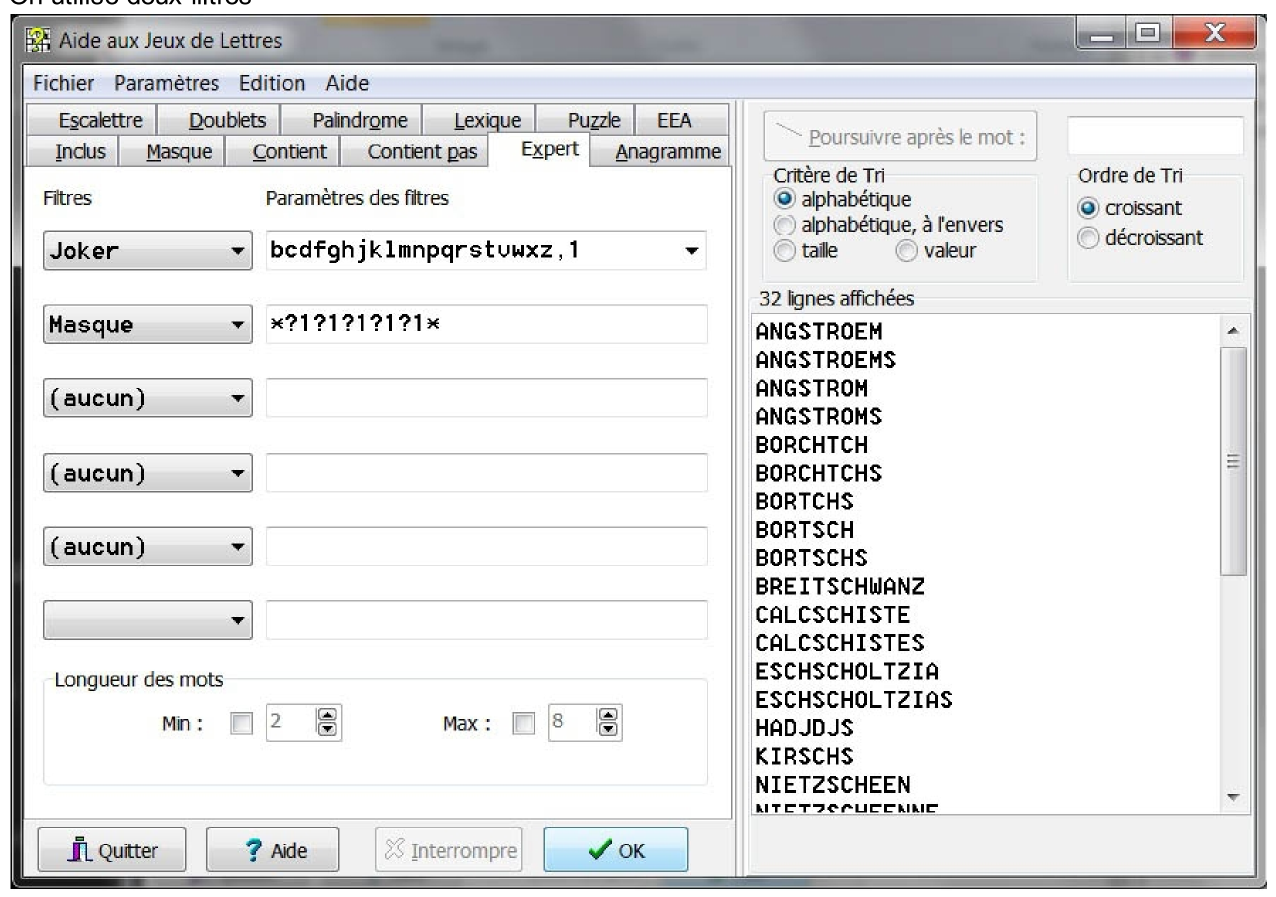
Recherche des mots possédant le plus grand nombre possible de voyelles

Piège !! On utilise ici la valeur des lettres.
1) Activez ce menu, appuyez sur le bouton zéro, pour mettre les compteurs à 0, puis sur les 6 touches A, E, I, O ,U, Y (ou cliquez sur les flèches associées à ces lettres) pour mettre à 1 les 6 valeurs des voyelles.
2)Utilisez le filtre "valeur min", que vous positionnez à une valeur suffisamment haute (8 par exemple)

C'est tout, et n'oubliez pas de trier la liste par valeur.

Recherche des mots qui contiennent 5 consonnes consécutives

On utilise deux filtres



Le premier filtre restreint aux consonnes le masque d'une lettre ?1. Le second filtre les mots qui commencent par n'importe quoi (premier joker *), puis 5 fois une lettre qui doit correspondre au joker ?1 (jokers ?1?1?1?1?1), puis n'importe quoi (dernier joker *).

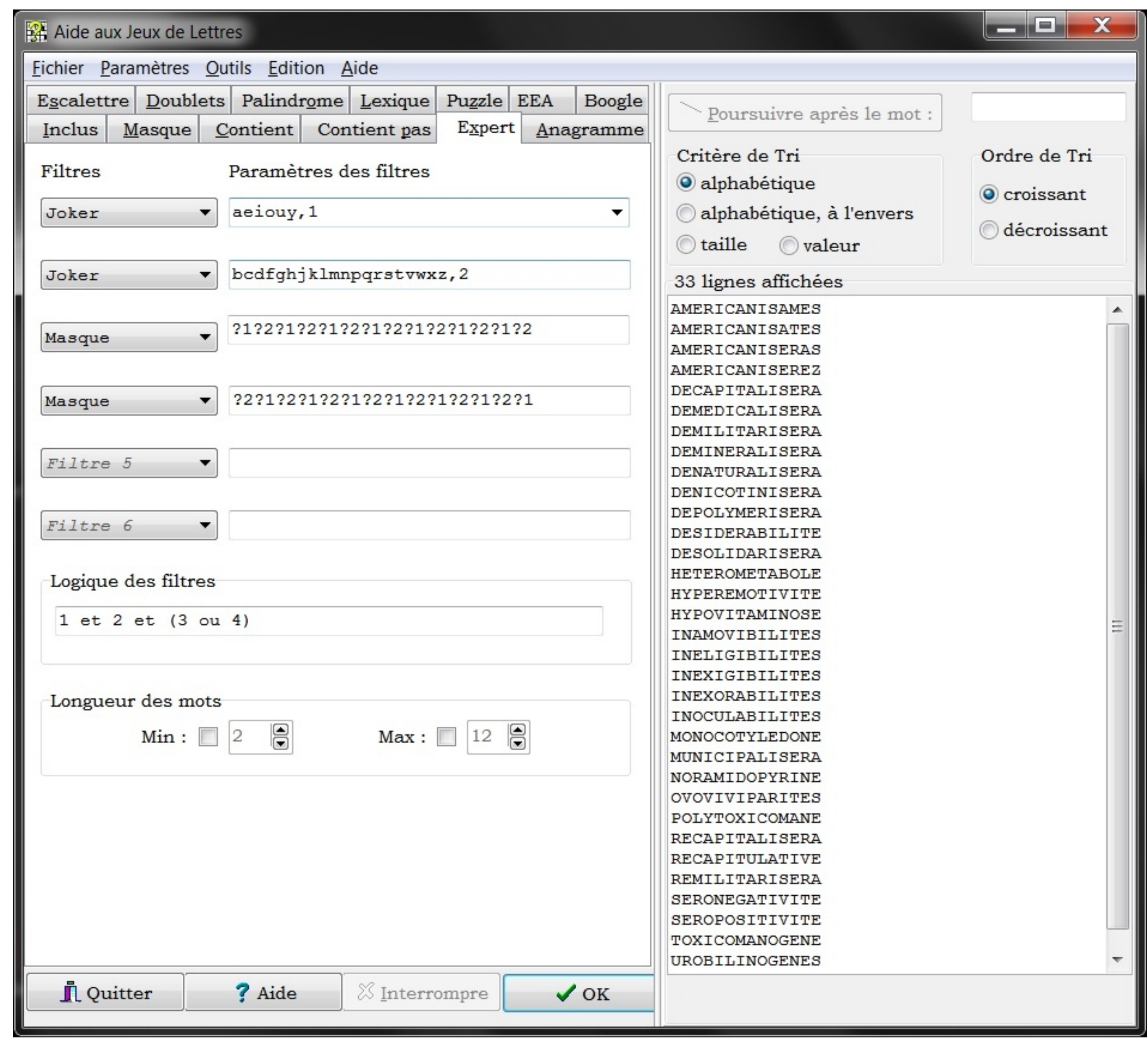
Recherche des hétérogrammes (mots ne possédant jamais deux fois la même lettre)

Il suffit d'utiliser l'onglet INCLUS !
Spécifiez "abcdefghijklmnopqrstuwxyz" comme jeu de lettres.

Recherche des mots ne contenant pas plus d'une consonne

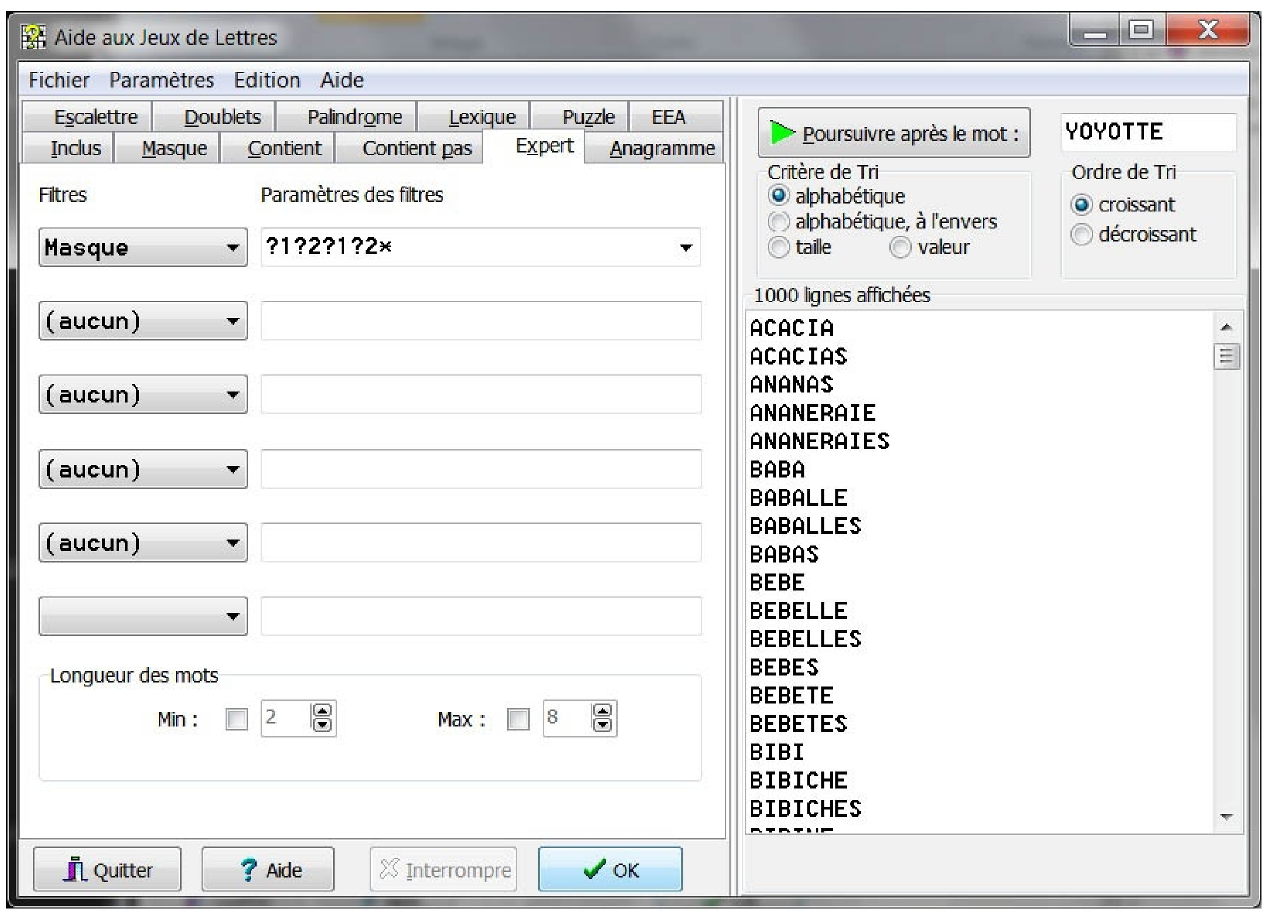
Il suffit d'utiliser l'onglet CONTIENT PAS
Spécifiez "bcdfghjklmnpqrstwxz" comme jeu de lettres, et indiquez que 1 des lettres du jeu de lettres est quand même tolérée. Remarque : l'onglet INCLUS convient moins bien, car il faudrait mettre la liste des consonnes et plusieurs (combien ?) exemplaires de chaque voyelle dans le jeu de lettres.

Recherche des mots de 14 lettres dont les lettres alternent voyelles et consonnes



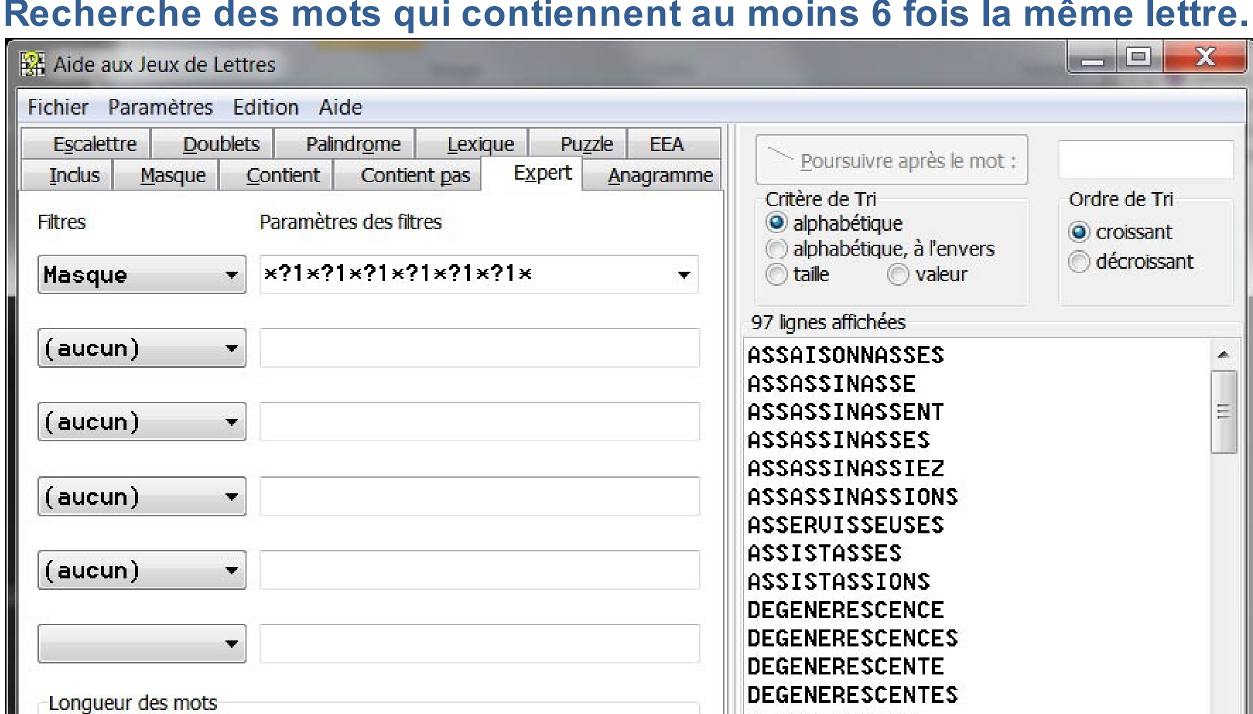
Le 1er filtre assigne les voyelles au joker 1, le 2ème les consonnes au joker 2. Le filtre 3 sélectionne les mots qui commencent par une voyelle, puis alternent consonnes/voyelles, et le filtre 4 sélectionne les mots qui commencent par une consonne puis alternent voyelle/consonne. On notera que **la logique des filtre a été modifiée** : on applique les filtres "**1 et 2 et (3 ou 4)**" car les mots recherchés satisfont l'un OU l'autre des filtres 3 et 4, mais pas les 2 à la fois ce qui est impossible.

Recherche des mots qui commencent 2 fois de suite par les mêmes deux lettres



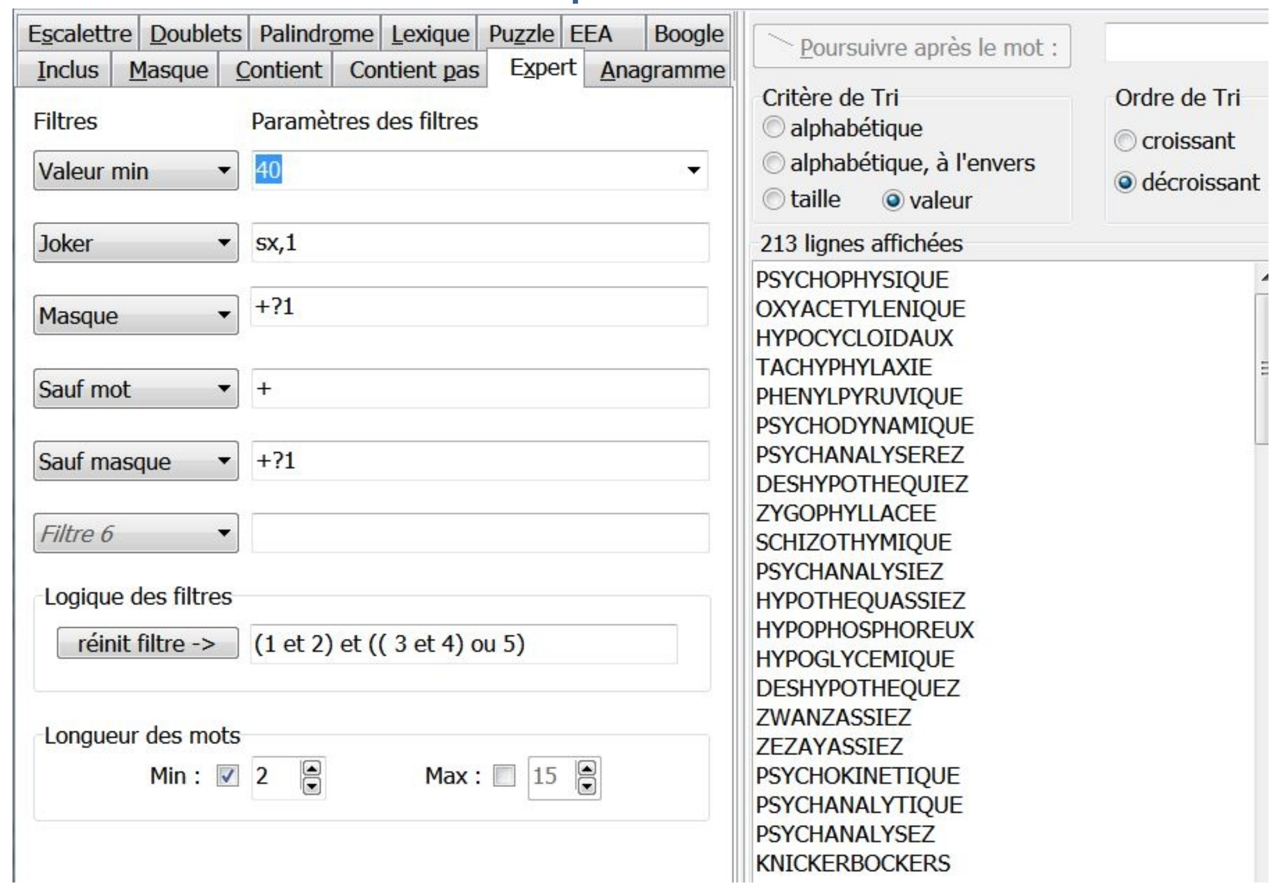
Remarque : le masque utilise deux fois chaque joker ?1 et ?2.
Cette fois, on choisira "**Un joker répété doit toujours masquer la même lettre""** dans le paramétrage du mode de fonctionnement des jokers, dans l'onglet Masque.

Recherche des mots qui contiennent au moins 6 fois la même lettre.



Le masque utilisé permet au mot de commencer par n'importe quoi (1er joker *), puis de continuer par une lettre (1er joker ?1), puis n'importe quoi (2ème joker *), puis la même lettre que celle du précédent joker ?1, etc... jusqu'au 6 ème joker ?1, qui peut être suivi de n'importe quoi (7ème joker *). Ce masque utilise 6 fois ?1 dont on veut qu'il masque toujours la même lettre. On choisira "**Un joker répété doit toujours masquer la même lettre**"" dans le paramétrage du mode de fonctionnement des jokers, dans l'onglet Masque.

Un exemple non trivial de logique de filtres
Filtrer les mots de plus de 40 points, en excluant les mots au pluriel.



La possibilité de modifier la logique des filtres permet de réaliser des recherches très fines. Par exemple, on désire ici sélectionner les mots de plus de 40 points au scrabble, en excluant leur version au pluriel (avec un s ou un x), pour ne conserver que la version au singulier. Plus précisément, le but est de ne pas afficher en double un mot et sa version au pluriel (exemple PSYCHOPHYSIQUE et PSYCHOPHYSIQUES).Dans un tel cas, on désire ne garder que la version au singulier PSYCHOPHYSIQUE. Si le mot n'existe qu'en version plurielle (exemple KNICKERBOCKERS), on conserve ce mot. Et on conserve aussi tous les mots qui ne sont pas des pluriels.

Explications du filtrage :
- le filtre 1 sélectionne les mots de plus de 40 points
- le filtre 2 ne filtre rien, mais restreint les 3 jokers numéro 1 (?1, *1, et +1) à ne masquer que les lettres S ou X
- le filtre 3 filtre les mots qui se terminent par le joker ?1, donc par la lettre S ou X. Au passage, le joker + est valorisé par tout ce qui précède ce S ou ce X terminal. Donc le + contient, potentiellement, la forme au singulier du mot.
- le filtre 4 vérifie si le +, donc le mot sans son S ou son X final, existe au dictionnaire. Si c'est le cas, le filtre Sauf Mot rejette le mot.

La succession logique des filtres 1 2 3 4 permet de retenir les mots de plus de 40 points, se terminant par un S ou par un X, et dont la version sans ce S ou ce X final n'est pas au dictionnaire.

- le filtre 5 teste si le mot ne correspond PAS au masque +?1, c'est à dire ne laisse passer que les mots qui ne se termine ni par S ni par X.

La succession logique des filtres 1 2 5 permet de retenir les mots de plus de 40 points, et ne se terminant ni par un S, ni par un X.

Au final, les mots que l'on souhaite afficher sont ceux qui correspondent à 1 et 2 et 3 et 4 , ou bien à 1 et 2 et 5. Le logique des filtres est donc :
(1 et 2 et 3 et 4) ou (1 et 2 et 5).
En mettant **1 et 2** en facteur, on peut aussi, au choix, écrire **(1 et 2) et ((3 et 4) ou 5).** Les deux écritures sont équivalentes.

retour Onglet Expert

Vous en voulez toujours plus? Alors, sachez que vous pouvez réaliser vos propres outils grâce à EEA.

retour Sommaire

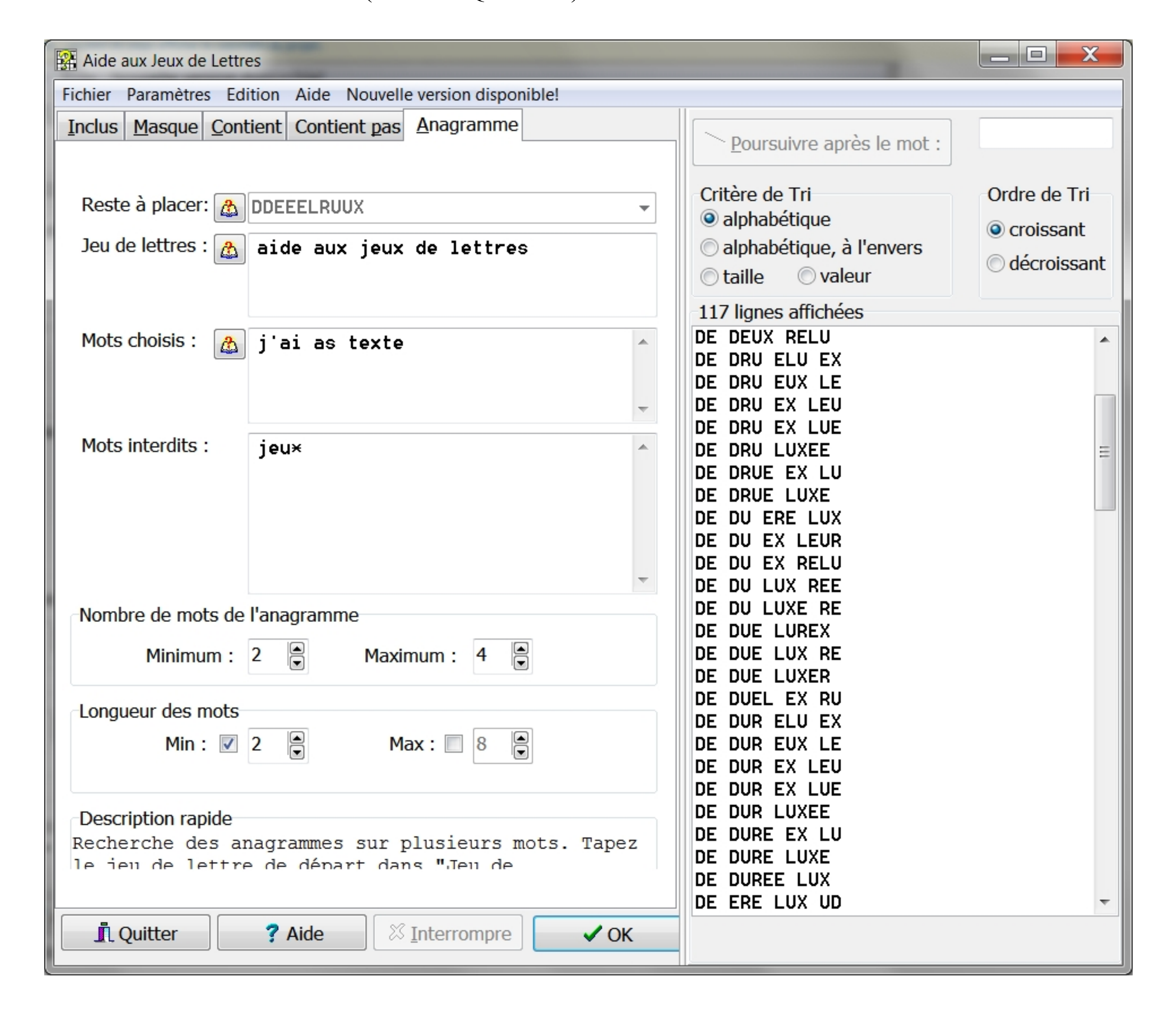
Onglet Anagramme

Onglet Anagramme

L'onglet "Inclus" permet déjà de rechercher les anagrammes d'un mot, c'est à dire les mots constructibles avec exactement les mêmes lettres. L'onglet Anagramme étend cette recherche aux anagrammes de plusieurs mots, ou d'une phrase entière. De plus, les caractères jokers '?' sont permis, ils remplacent une (une seule) lettre quelconque.

Exemple : un anagramme de "Aide aux jeux de lettres" découvert avec cette fonction :

- "J'ai relu deux as de texte" (Perec et Queneau?)



Mode d'emploi:

1) Tapez votre texte d'origine dans la boîte intitulée "Jeu de lettres". Vous constatez que les lettres du texte s'ordonnent toutes seules dans la boîte "Reste à placer". N'effectuez donc pas de saisie directe dans cette boîte. Vous pouvez utiliser autant de jokers '?' (qui représentent une lettre quelconque) que vous le désirez.

2) Choisissez le nombre maximum de mots que devra comporter l'anagramme
3) Appuyez sur "OK" !

Probablement, vous aurez beaucoup de possibilités, surtout si la phrase d'origine est longue et si vous

avez utilisé des jokers '?'.

Mots choisis

Essayez de choisir à "priori" un ou plusieurs mots que vous voulez voir apparaître dans l'anagramme,surtout si vous avez des lettres délicates à placer. Faites cela avec les autres onglets d'AJL (les lettres dans "lettres restantes" peuvent être directement récupérées dans "Expert" ou "Inclus", à priori les onglets que vous utiliserez)

Tapez ces mots imposés dans la boîte "mots choisis" (ou faites un drag and drop) : vous constatez que les lettres de ces mots sont automatiquement retirées de la boîte "lettres restantes". Les lettres jokers '?' sont consommées autant que nécessaire si vous imposez des mots impossibles à réaliser compte-tenu des lettres encore disponibles.

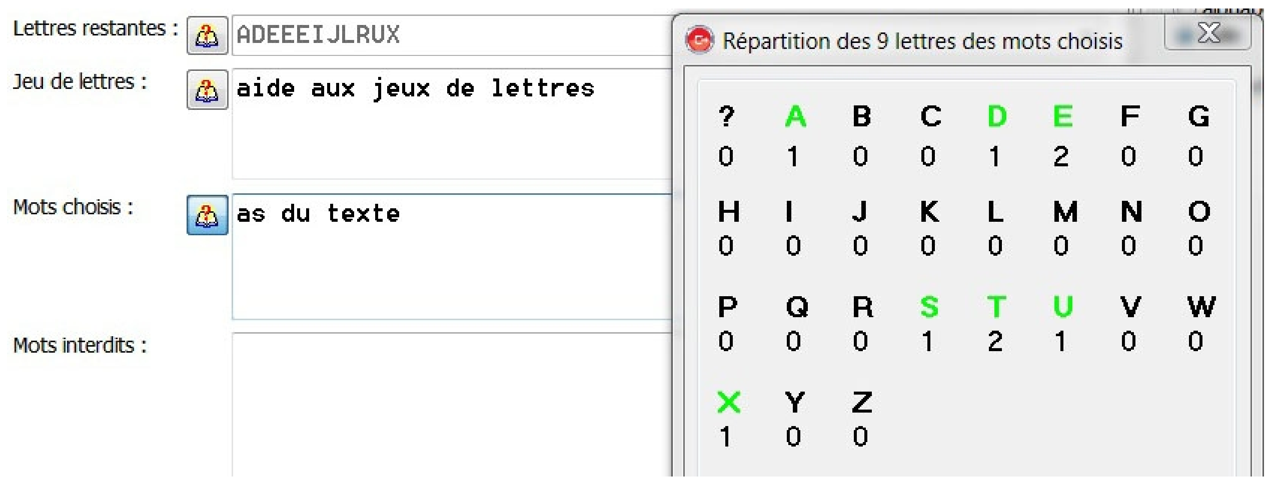
Mots interdits

Il est fréquent que certains mots ou ensembles de mots que vous ne souhaitez pas retenir, viennent polluer les anagrammes proposés. Pour les éviter, tapez-les (ou faites un drag and drop) dans la boîte « mots interdits », en les séparant par des blancs ou des virgules. Vous pouvez utiliser toute la puissance des caractères génériques *,+, ? pour interdire tous les mots correspondant à un masque. Par exemple LIVR* pour éliminer les mots de racine LIVRE, ou *AIENT si vous voulez éviter que ces précieuses voyelles soient utilisés par cette conjugaison.

Continuez ainsi en affinant les interdits et les choisis jusqu'à obtention des solutions souhaitées.

Les 3 boutons ornés d'un livre





Appuyez sur un des 3 boutons en forme de livre ornés d'un point d'interrogation pour afficher une nouvelle fenêtre permettant d'obtenir la répartition des lettres présentes dans les 3 zones de texte :

Pour la zone de texte "Jeu de lettres", cette répartition donne le nombre de chacune des lettres présentes, y compris les jokers '?' s'ils sont utilisés. Même chose pour la zone "Mots choisis"
En ce qui concerne la zone lettres restantes, les lettres surconsommées (ie : présentes en plus grand nombre dans les "mots choisis" que dans le "jeu de lettres", même en tenant compte des jokers) sont indiquées en rouge. A priori, ce sont des erreurs de votre part !

Une seule fenêtre est affichée à la fois, correspondant à celui des 3 boutons qui est enfoncé. Pour fermer cette fenêtre, appuyer à nouveau sur le bouton enfoncé.

Remarque :

L'option de coloration des mots selon le dictionnaire de provenance n'est pas supportée dans cette fonction.

retour Mode d'emploi

Onglet Palindrome

Onglet Palindrome

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert".

Cet onglet aide à la réalisation de palindromes, c'est-à-dire de textes qui restent identiques qu'ils soient lus de gauche à droite ou de droite à gauche, à l'instar du célèbre "**ESOPE RESTE ICI ET SE REPOSE**". En effet, cette phrase présente la même succession de lettres dans les 2 sens de lecture.

Les deux onglets en miroir.

Pour vous aider à accomplir cette difficile tâche, AJL présente 2 onglets de saisie, intitulés "**de droite à gauche**" et "**de gauche à droite**". Ces deux onglets sont miroir l'un de l'autre : tout texte saisi, tout caractère ajouté ou soustrait dans l'un des deux onglets est reproduit à l'envers dans l'autre... Sauf les caractères "espace", qui sont ignorés, ce qui vous permet de les utiliser pour séparer les mots de manière indépendante dans les 2 onglets. Vous utiliserez alternativement chacun des deux onglets pour affiner votre palindrome.

Les boutons Défaire et Refaire.

Chaque essai de palindrome effectué par appui sur le bouton Ok provoque l'enregistrement intégral du texte dans une sauvegarde interne, capable de conserver les 8 derniers essais. A tout instant, vous pouvez revenir sur un de ces 8 derniers essais en appuyant sur le bouton Défaire. Si vous allez trop loin en arrière,vous pouvez repartir dans l'autre sens avec le bouton Refaire. Vous apprécierez la facilité apportée par ces boutons car la construction d'un palindrome est essentiellement basée sur une démarche essai-erreur.

Le rôle du caractère joker *

Prenons l'exemple du palindrome "ESOPE etc...". Si on débute un palindrome par le mot ESOPE, il en découle qu'il se termine par EPOSE. Si on remplace les caractères manquants par*, le palindrome a donc la structure suivante :

- dans le sens **gauche droite** "ESOPE *"
- dans le sens **droite gauche** "*EPOSE"

Notez bien l'espace séparateur de mot entre ESOPE et *, car ESOPE est un mot complet. Ce n'est pas le cas pour EPOSE, et donc * colle à ce mot dans le sens droite gauche. Le caractère *, présent dans les 2 cadres, symbolise la suite du palindrome. Il doit être séparé des caractères par un espace quand ces caractères forment un texte cohérent, il reste collé au dernier groupe de caractères si ceux-ci ne sont qu'une portion de mot.

Ce que AJL fait pour vous

Commencez par choisir l'onglet (gauche-droite ou droite-gauche) que vous voulez compléter. Tout simplement, AJL va considérer * comme un joker et chercher les mots qui respectent la contrainte de palindrome. Dans notre exemple, si vous travaillez sur "*EPOSE", AJL cherchera tous les mots finissant par EPOSE. Par exemple REPOSE, et aussi ENTREPOSE. Mais AJL réfléchit aussi au coup suivant. Considérant REPOSE, l'ajout du R transforme, dans le cadre gauche-droite, le texte "ESOPE *" en "ESOPE R*". AJL vérifie alors l'existence de mots commençant par R pour accepter le mot REPOSE, ce qui est le cas. Considérant ENTREPOSE, "ESOPE *" devient "ESOPE RTNE*". Il n'y a aucun mot

commençant par RTNE. AJL ne vous proposera donc pas le mot ENTREPOSE pour compléter le palindrome. De plus, AJL signale en ***gras et italique*** les mots dont l'ajout est susceptible de terminer votre palindrome (sans préjuger du sens de la phrase, naturellement).

Et plus encore : le drag and drop, ou "glisser-déplacer"

Double-cliquez sur un mot de la liste de mot, par exemple REPOSE, en maintenant le bouton enfoncé au deuxième clic. Toujours bouton enfoncé, déplacez le curseur dans le cadre qui vous a servi, par exemple celui qui contenait "*EPOSE". Relâchez. Hop ! "*EPOSE" devient "* REPOSE". Vous pouvez donc travailler vos palindromes presque entièrement à la souris, sans rien taper au clavier, sinon les espaces entre les mots. Un palindrome est obtenu dès que les deux textes, mis à la suite l'un de l'autre sans tenir compte des 2 *, forment une phrase correcte.

retour Mode d'emploi

Onglet Doublets

Onglet Doublets

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert".

Cet Onglet propose trois fonctions complémentaires :

1) "Chercher un chemin du mot initial à la cible"

Il s'agit de trouver une liste de mots , du plus petit nombre nombre possible, qui permette de transformer progressivement le mot initial en un mot cible, en ne changeant qu'une seule lettre de chaque mot.
Les paires de mots qui peuvent être ainsi reliées sont appelées doublets de Carroll, d'où le nom de l'onglet.

Exemple
On peut relier NOIR à VERT avec: NOIR - **S**OIR - SOI**T** - SO**R**T - S**E**RT - **V**ERT.

2) "Chercher les mots éloignés du mot initial de n

Il s'agit de rechercher les mots que l'on ne peut relier au mot initial qu'avec n mots intermédiaires, si on respecte la contrainte que chaque mot intermédiaire ne diffère que d'une lettre de son prédécesseur.

Exemple
Les mots suivants sont éloignés de 1 seul mot (dans ce cas, celà signifie différents d'une seule lettre) de "JEU" :
FEU, HEU, JET, LEU et PEU.

3) "Chercher les mots le plus éloignés du mot initial

Il s'agit de rechercher les mots reliés au mot initial par le plus long chemin irréductible possible, toujours avec la contrainte que chaque mot intermédiaire ne diffère que d'une lettre de son prédécesseur.

Exemple
En partant du mot "COQ", on peut joindre "OXO" par le chemin COQ - COU - POU - PLU - ELU - ECU - ECO - EXO - OXO
L'algorithme AJL vous garantit qu'aucun chemin plus court n'existe entre ces deux mots, et qu'aucun mot n'est plus éloigné de COQ que OXO.

Caractères génériques

Les caractères génériques ne sont pas supportés par cette fonction

Astuce : vous pouvez transférer un mot de la liste dans la zone d'édition "mot cible" par un glisser-déplacer. Il suffit de double-cliquer sur le mot choisi de la liste en laissant le bouton gauche de la souris enfoncé au deuxième clic. Vous pouvez alors "glisser" le mot, toujours bouton gauche enfonçé, vers la zone d'édition choisie. Le mot se dépose dès que vous relâchez le bouton.

[retour Mode d'emploi](#)

Onglet Escalettre

Onglet Escalettre

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert".

L'onglet Escalettre permet de trouver des suites de mots satisfaisant à la contrainte suivante : chaque mot utilise toutes les lettres du mot précédent, plus une, dans un ordre quelconque.
Pour celà trois modes de fonctionnement sont proposés selon le radio-bouton séllctionné.

1) "Chercher un chemin du mot initial à la cible"

Ce choix permet de trouver une des suites de mots qui relient un mot initial à un mot cible. AJL n'affiche que l'une des possibilités, vous pouvez trouver des variantes en recherchant les anagrammes des mots proposés.

Exemple de ARTISTE à TRAVESTISSEMENT

ARTISTE
ARIETTES
RAINETTES
ASTREINTES
MARTENSITES
TARISSEMENTS
TRANSVESTISME
AVERTISSEMENTS
TRAVESTISSEMENT

2) "Chercher les mots éloignés du mot initial de n"

L'onglet permet de trouver les mots situé à une certaine "distance" d'un mot donné (dans l'exemple plus haut, on dit que ARIETTES est à une distance 2 de ASTREINTES, car il faut ajouter deux lettres.

3) "Chercher les mots le plus éloignés du mot initial

Il s'agit de rechercher les mots reliés au mot initial par le plus long chemin irréductible possible, toujours avec la contrainte que chaque mot intermédiaire ajoute une seule lettre à son prédécesseur.
L'algorithme AJL vous garantit qu'aucun chemin plus court n'existe entre ces deux mots, et qu'aucun mot n'est plus éloigné du mot initial choisi.

Remarque

Astuce : vous pouvez transférer un mot de la liste dans la zone d'édition "mot cible" par un glisser-déplacer. Il suffit de double-cliquer sur le mot choisi de la liste en laissant le bouton gauche de la souris enfoncé au deuxième clic. Vous pouvez alors "glisser" le mot, toujours bouton gauche enfoncé, vers la zone d'édition choisie. Le mot se dépose dès que vous relâchez le bouton.

[retour Mode d'emploi](#)

Onglet Puzzle

Onglet Puzzle

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert".

Cet onglet recherche comment assembler des fragments de mots pour constituer un ou plusieurs mots, un peu comme si chaque fragment de mot était une pièce d'un puzzle, et qu'il faille retrouver une phrase avec ces pièces. Il est donc très semblable à l'onglet Anagramme, qui travaille de manière tout à fait similaire, mais avec des fragments d'une et une seule lettre. Une autre différence est que cet onglet n'accepte pas l'utilisation de caractères jokers.

Vous précisez les pièces du puzzle en les tapant dans le cadre prévu, et en veillant à les séparer les unes des autres par des virgules ou des blancs.

Exemple

Avec les pièces de puzzle suivantes :
a, u, t , o, mat , i , qu, e

et en spécifiant dans les choix possibles des solutions à exactement deux mots de taille quelconque, on trouve :
AI MAT TOUQUE
AMATI TOUQUE
AUTOMATE QUI
MATAI TOUQUE
MATOU QUETAI
MATTEAU QUOI

il est intéressant de constater que la fonction Anagramme trouve quant à elle 35 solutions dans les mêmes conditions, certaines utilisant le mot at omi que dont il est facile de voir qu'il est impossible à construire avec les fragments de puzzle choisis.

retour Mode d'emploi

Onglet Lexique

Onglet Lexique

Attention cet Onglet n'est affiché qu'en mode expert. Voyez ici comment passer du mode "débutant" au mode "expert"..

Cet onglet recherche comment utiliser des fragments de mots pour constituer un mot, un peu comme si chaque fragment de mot était un nouvel alphabet. En ce sens, il généralise l'onglet Inclus, dont le lexique est un sous-ensemble des 26 fragments d'une seule lettre correspondant à l'alphabet.

Il ne faut pas le confondre avec l'onglet Puzzle, qui travaille aussi avec des fragments de mots mais de façon très différente :

- il n'est pas obligatoire d'utiliser tous les éléments du lexique dans la construction des mots recherchés (alors que Puzzle emploie une et une seule fois chaque fragment)
- les éléments du lexique peuvent être réemployés autant de fois que nécessaire (alors que Puzzle les emploie une et une seule fois)
- Lexique cherche à assembler un et un seul mot à la fois, en choisissant les fragments utiles.
- Lexique sait utiliser l'option de colorations des mots selon le dictionnaire de provenance.

Exemple

Avec le lexique suivant (correspondant à certains codes de pays européens, utilisés dans les plaques

minéralogiques de véhicules) :
a, b, ch, d, e, f , gb, h, i , p, pl

on peut former le mot suivant, le plus long possible:
PI EDDEBI CHE

retour Mode d'emploi

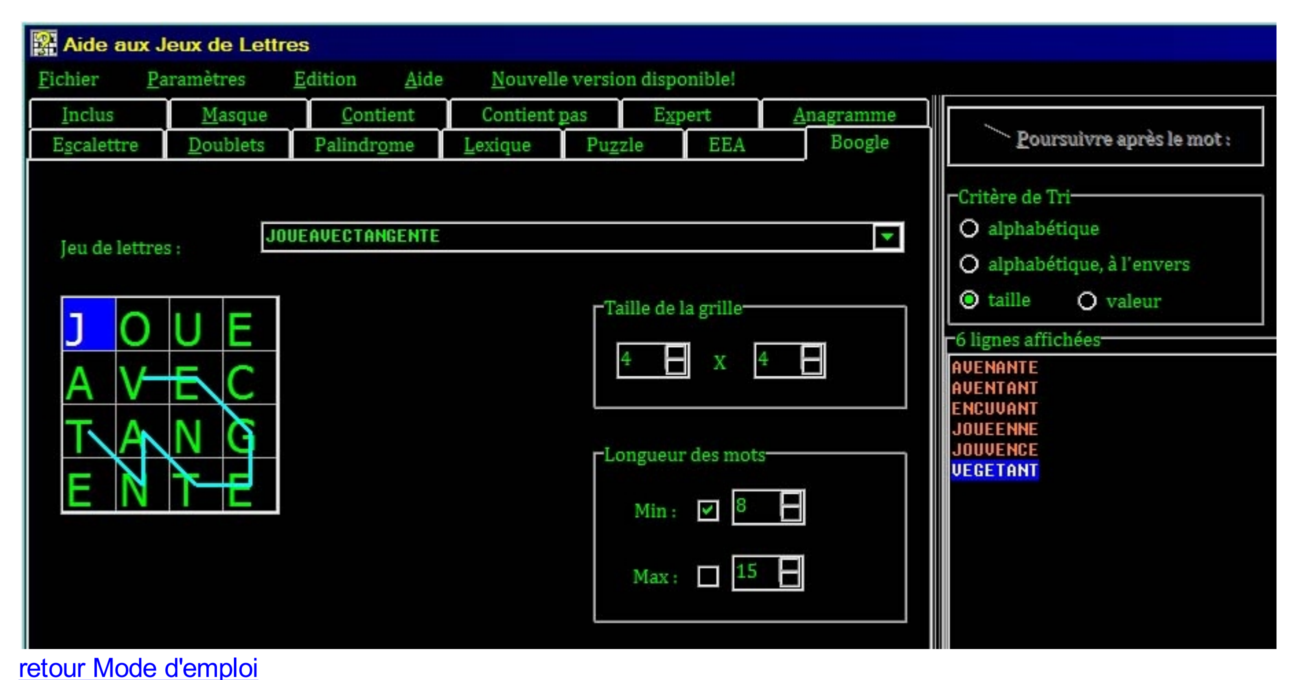
Onglet Boogle

Onglet Boogle

Cet onglet recherche les mots constructibles à l'aide d'une grille carrée contenant une lettre dans chaque case. Les lettres successives des mots éligibles doivent être adjacents dans la grille, c'est à dire que chaque lettre doit "toucher" la suivante dans la grille, verticalement, horizontalement ou en diagonale. Une lettre de la grille ne peut être utilisée qu'une seule fois dans un même mot.

Les lettres de la grille sont simplement entrées dans le champ texte "Jeu de lettres" et AJL s'occupe de les ranger dans l'ordre dans la grille. Vous disposez du choix des dimensions de la grille : le nombre de colonnes ou de lignes peut varier entre 3 et 8.

Complément:
Si vous double-cliquez sur un mot de la liste des mots trouvés, les lettres de la grille qui lui correspondent sont reliés par un trait, comme dans l'illustration ci dessous.



retour Mode d'emploi

Onglet EEA

Onglet EEA

Attention cet Onglet n'est affiché qu'en mode expert.
Voyez ici comment passer du mode "débutant" au mode "expert".

Cet onglet permet la rédaction, le test et la sauvegarde de programmes écrits en langage EEA.

Accès direct au langage EEA

Voyez la description générale de EEA pour apprendre à vous servir de ce langage.
Voyez la liste des fonctions et opérateurs .

L'onglet 'Evaluation d'Expressions AJL se compose de haut en bas :

1) d'un cadre permettant l'édition de vos programmes EEA.
2) d'une ligne horizontale de redimensionnement, juste entre les deux cadres, qui peut être déplacée à la souris.
3) d'un cadre d'affichage de messages, émis automatiquement ou volontairement dans votre programme par l'instruction Message.

Tracer, Trier

Tracer les variables   Trier

le mode Trace

Au dessus de l'habituelle liste des mots,on trouve une case à cocher "Tracer les variables". Si elle est activée, toute exécution d'une instruction d'affectation provoque un message : [Trace] variable = valeur. Vous pouvez modifier cette case à cocher même au beau milieu de l'exécution d'un script. Vous pouvez aussi utiliser l'instruction Trace dans vos scripts.

Trier la zone Sélection

Si le tri des mots sélectionnés dans tous les autres onglets a un sens et une utilité, il n'en est pas de même pour EEA, puisque la zone des mots sélectionnés va contenir seulement le résultat des instructions Selection exécutés par le script. C'est pourquoi une case à cocher, inactivée par défaut, est à votre disposition pour éventuellement trier la zone Sélection. Cette action de tri n'est effectuée qu'à la fin (ou l'interruption) du script EEA exécuté.

Edition de script EEA : 4 trucs à savoir.



Entre la zone d'édition et la zone message on trouve 4 boutons dont la signification est la suivante:

- **bouton //** (ou raccourci clavier CTRL + /)

Ce bouton ajoute les deux barres de commentaire // en début d'une ligne (celle du curseur) ou de plusieurs lignes (si sélectionnées). Si les deux barres // sont déjà présentes en début de ligne, l'action réalisée est alors le retrait de ces deux barres. Ce bouton échange donc une ligne entre un status instruction(`instruction`) et un status commentaire (`//instruction`)

- **bouton <<** (ou raccourci clavier CTRL + flèche gauche)

Ce bouton supprime deux espaces en début de ligne (celle du curseur) ou de plusieurs lignes (si sélectionnées). Ceci est très pratique pour gérer les indentations optionnelles des instructions à l'intérieur d'un couple d'instruction bloc.

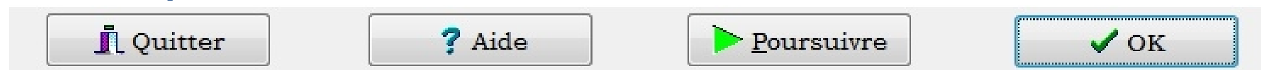
- **bouton >>** (ou raccourci clavier CTRL + flèche droite)

Ce bouton ajoute deux espaces en début de ligne (celle du curseur) ou de plusieurs lignes (si sélectionnées). Ceci est très pratique pour gérer les indentations optionnelles des instructions à l'intérieur d'un couple d'instruction bloc.

- **bouton Aller ligne** (ou raccourci clavier "Entrée" dans le champ de saisie à sa droite)

Ce bouton déplace le curseur à la ligne dont le numéro est saisi juste à droite. Un appui sur la touche entrée après saisie du numéro de ligne a le même effet.

Interrompre, Poursuivre



Habituellement grisé, le **bouton Interrompre** est actif pendant le déroulement d'un script EEA. Si vous appuyez sur ce bouton, le script s'interrompt immédiatement. Si le script a été interrompu dans des conditions qui permettent sa poursuite ultérieure, le **bouton Poursuivre** apparaît. Pressez le pour continuer le script exactement de là où il en était. Lors de l'interruption, le contenu de toutes les variables est affiché dans la zone message si vous avez activé l'option correspondante dans Paramètres -> Options générales -> EEA

retour Mode d'emploi

retour Introduction

Dictionnaires

Les dictionnaires

Pour AJL, un dictionnaire est :

- soit un fichier fourni par JLP fractware, d'extension ".AJL" ou ".PJL",
- soit un fichier que vous fournissez vous même, un simple fichier texte d'extension ".TXT".

Ces dictionnaires se rangent en trois catégories :

1) les dictionnaires alphabétiques tout en majuscules

C'est le cas général de ceux proposés en téléchargement sur le site JLP fractware. Ils sont en extension ".AJL". Par exemple "SCRABBLE.AJL". Quand AJL manipule un tel dictionnaire, il passe automatiquement en mode "majuscules seules". Vous pouvez taper vos recherches en minuscules ou en majuscules, peu importe. AJL passe tout en majuscules, pour rester cohérent avec le dictionnaire chargé.
Vous pouvez parfaitement construire vos propres dictionnaires de ce type. Il suffit de respecter les quatre consignes suivantes :

- un seul mot par ligne
- en majuscules sans accents, ni espaces, trémas, tirets, cédilles etc.. Que des majuscules !
- ranger chaque mot dans l'ordre alphabétique.
- enregistrer le fichier avec l'extension ".txt", sous forme de fichier texte. Utilisez pour celà un éditeur de texte simple comme Notepad par exemple.

Lors de la lecture des dictionnaires choisis, les doublons sont automatiquement éliminés, n'hésitez donc pas à utiliser vos propres fichiers de mots en plus de ceux fournis avec AJL.

2) les dictionnaires alphabétiques utilisant des minuscules ou des accents

Par exemple le dictionnaire "minucules.ajl", proposé sur le site. Il contient 125153 mots avec accents et minuscules. Quand AJL manipule un tel dictionnaire, il passe automatiquement en mode de support des minuscules et majuscules. Ce que vous tapez dans les onglets est considéré tel qu'il est tapé, et les lettres minuscules, majuscules, avec accents sont autant de lettres différentes. Par exemple, vous ne trouverez pas le mot "livre" en cherchant les mots qui contiennent "L", ni le mot "LIVRE" en cherchant ceux qui contiennent 'e'.
Vous pouvez parfaitement construire vos propres dictionnaires de ce type. Il suffit de respecter les trois consignes suivantes :

- un seul mot par ligne
- ranger chaque mot dans l'ordre. Attention cet ordre n'est pas naturel, il faut utiliser, pour le tri, un utilitaire informatique. Mais AJL peut le faire pour vous, voyez plus bas.
- enregistrer le fichier avec l'extension ".txt", sous forme de fichier texte. Utilisez pour celà un éditeur de texte simple comme Notepad ou bien mieux l'excellent UltraEdit par exemple.

3) les dictionnaires phonétiques

Il n'y en a qu'un seul à ce jour, "phonetique.pjl", proposé sur le site. Il contient des mots en représentation phonétique, laquelle utilise des lettres minuscules, majuscules, et avec accents. Quand AJL manipule un tel dictionnaire, il passe aussi en mode de support des minuscules et majuscules. Vous ne pouvez pas, à ce jour, construire vos propres dictionnaires phonétiques.

Les outils dictionnaires

AJL propose trois outils permettant de faciliter la tâche de création de vos propres dictionnaires. Ils sont disponibles dans le menu Outils.
On y trouve les outils pour :

Création de dictionnaire
Comparaison de fichiers
Extraction de mots

Astuce

Il est possible d'afficher la liste des caractères utilisés dans le (ou les) dictionnaires chargés. Il suffit d'aller

dans l'onglet "Anagramme" et d'appuyer sur un des trois petites boutons 



Menus

Les menus

Fichier

Enregistrer sous:
Permet de sauvegarder dans un fichier la liste de mots trouvés et affichés par AJL

Dictionnaires alphabétiques : chargement en mémoire de nouveaux dictionnaires alphabétiques (type AJL par exemple)
voir chargement de dictionnaire

Dictionnaires phonétique : chargement de dictionnaire phonétique, comme PHONETIQUE.PJL
voir symboles phonétiques
voir chargement de dictionnaire

Nouveau script EEA
Effacement du script en cours d'édition, de façon à composer un nouveau script EEA.

Ouvrir script EEA
Lecture d'un fichier .EEA contenant un script à exécuter.

Réouvrir script EEA
Ce sous-menu vous propose de choisir parmi les 10 derniers scripts EEA ouverts.

Enregistrer script EEA
Sauvegarde d'un fichier .EEA sous le même nom que celui en cours d'utilisation (affiché dans le titre de la fenêtre)

Enregistrer script EEA sous...
Sauvegarde d'un fichier .EEA sous un autre nom que celui en cours d'utilisation.

NB : ces 5 lignes de menu n'apparaissent qu'en mode expert, et si l'onglet EEA est celui ouvert.

Imprimer
Imprime la liste des mots trouvés et affichés. AJL ajuste automatiquement le nombre de colonnes nécessaires en fonction de la largeur du papier et de la longueur des mots à imprimer.

Quitter
Quitter le programme AJL, quel dommage.

Paramètres

Paramètres -> Valeur des lettres...
Paramètres -> Symboles phonétiques...
Paramètres -> Options générales...

Outils

Nouveau Dictionnaire...
Outil d'aide à la création d'un dictionnaire à partir d'une liste "brute" de mots.

Comparer fichiers...
Outil de comparaison de fichiers listes de mots

Extraire les mots...
Outil de fabrication d'une liste "brute" de mots, à partir d'un texte.

NB : ces 3 lignes de menu n'apparaissent qu'en mode expert.

Edition

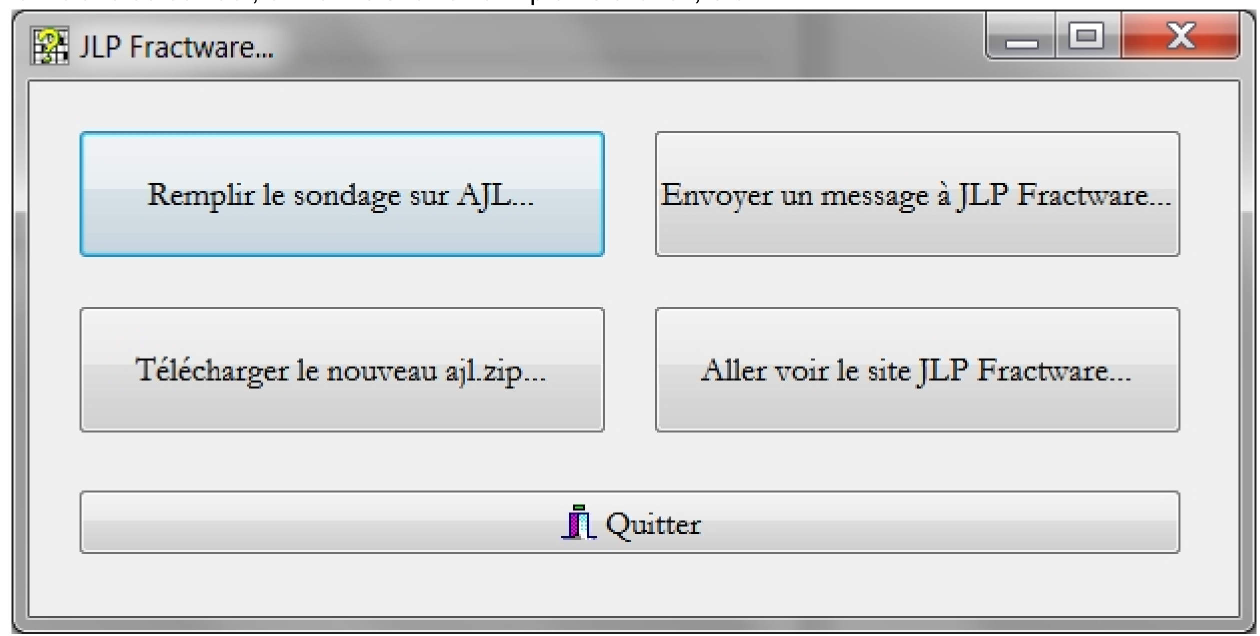
Copier
Permet de transférer dans le presse papier standard de Windows les mots sélectionnés.

Tout copier
Sélectionne et transfère toute la liste de mots dans le presse papier de Windows

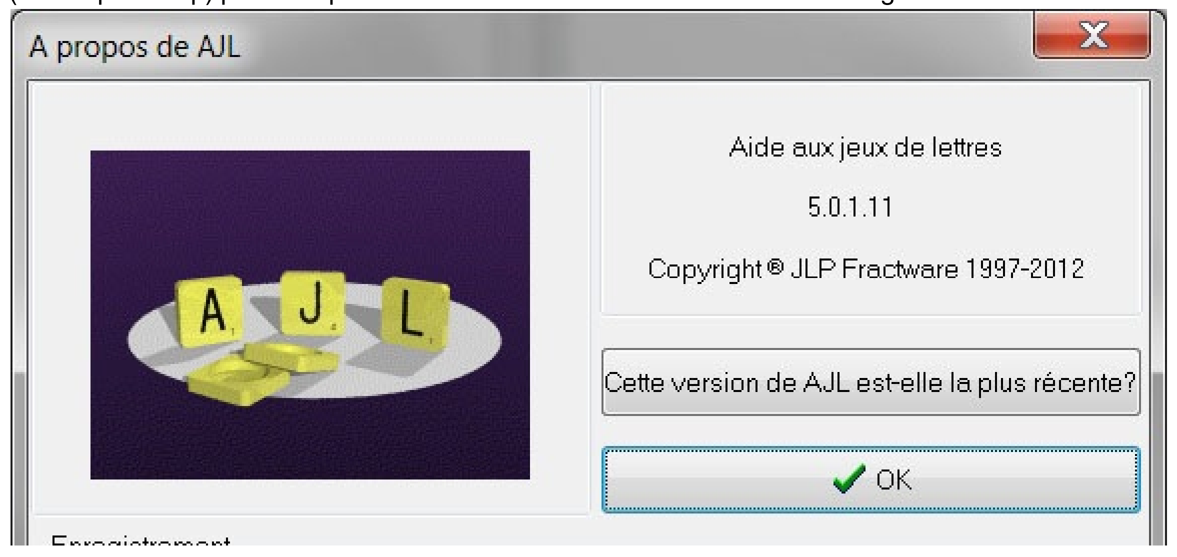
Coller (CTRL + V)
Copie le contenu du presse papier dans le champ de saisie principal d'AJL.

Aide

JLP Fractware
Affiche une série de boutons qui renvoient vers le site internet JLP Fractware, sur une page de sondage, un formulaire de contact, un lien vers le fichier zip d'installation, etc...



A propos..
Affiche une fenêtre d'informations où vous trouverez notamment le numéro de la version courante d'AJL deux œufs de Pâques, et un bouton intitulé "**Cette version est elle la plus récente** ?" qui interroge directement mon site (une requête http) pour récupérer le numéro de la dernière version mise en ligne.



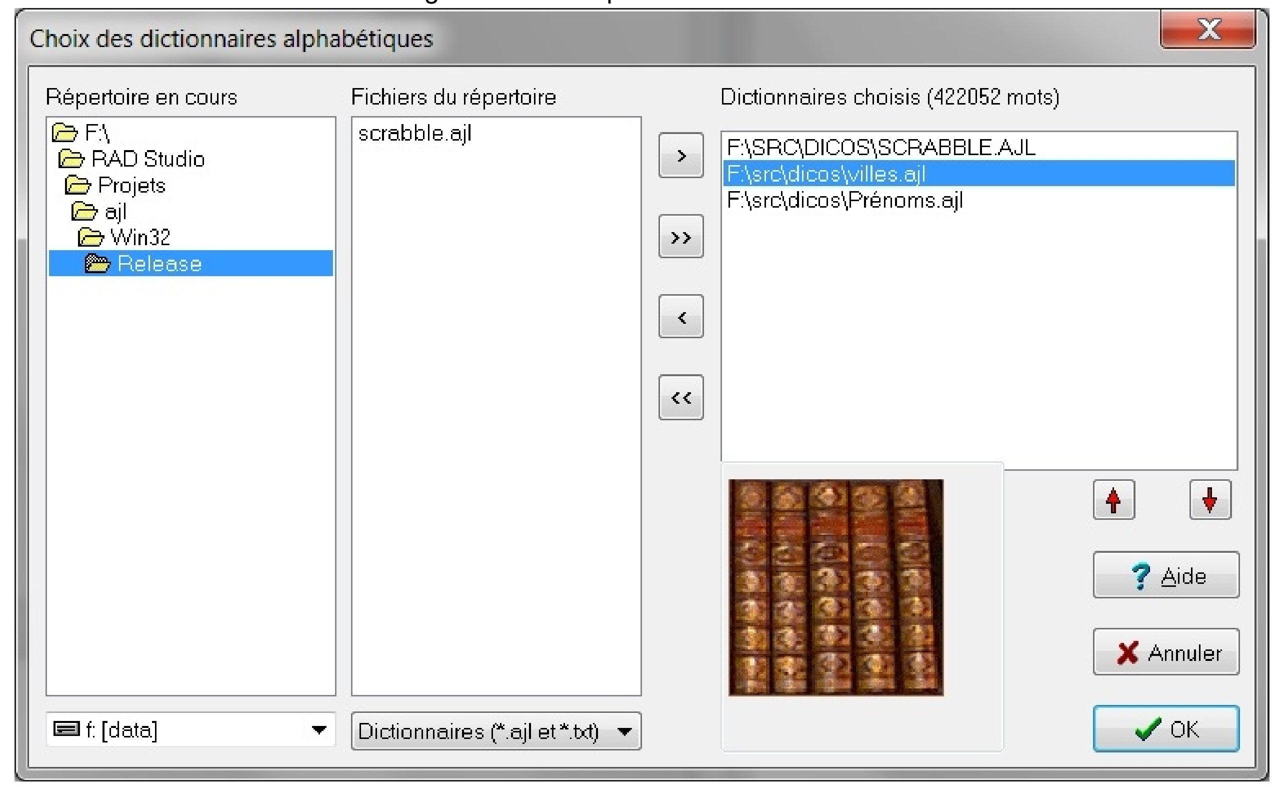
Nouvelle version disponible

Il ne s'agit pas vraiment d'un menu mais d'une commande qui apparaît automatiquement si votre version de AJL n'est pas la dernière version mise en ligne. L'appui sur cette pseudo-entrée de menu lance simplement la même fenêtre que celle disponible dans l'aide à la rubrique "JLP Fractware...". Aucune installation automatique n'est réalisée.

Chargement de dictionnaire

Chargement de dictionnaire

Fonctionnement de la boîte de dialogue accessible par le menu **Fichier->Dictionnaires**



Elle permet de changer le ou les dictionnaires de référence utilisés, y compris ceux que vous aurez créés vous même. Voir fin de cette page.

AJL affiche alors une boîte de dialogue de recherche de fichiers (les deux cadres de gauche), et précise dans une liste affichée à droite de la fenêtre la liste des dictionnaires en déjà sélectionnés.

- **Le bouton >**

ajoute le ou les fichiers choisis à la liste des dictionnaires.   Vous pouvez aussi pratiquer le glisser-déplacer depuis la liste des fichiers vers la liste des dictionnaires. Un double-clic sur un fichier a le même effet.

- **Le bouton >>**

ajoute tous les fichiers affichés à la liste de dictionnaires.

- **le bouton <**

retire le ou les dictionnaires sélectionnés de la liste des dictionnaires. (Cette action ne détruit aucun fichier!) Vous pouvez aussi pratiquer le glisser-déplacer depuis la liste des dictionnaires vers la liste des fichiers. Un double clic sur un des dictionnaires choisis a le même effet.

- **le bouton <<**

efface entièrement la liste des dictionnaires sélectionnés. Cette action ne détruit aucun fichier.

- **les boutons "fléche en haut"  et "flèche en bas"**

localisés juste au dessous de la liste des dictionnaires choisis ne servent que lorsque vous voulez contrôler précisément l'ordre dans lequel les dictionnaires sont traités. Sélectionnez un (ou plusieurs) dictionnaire, et modifiez avec ces boutons son emplacement dans la liste. Ceci peut s'avérer utile quand vous replacez un des dictionnaires de la liste, et désirez préserver les couleurs affectées aux autres dictionnaires. Dans un tel cas, il vous suffit de déplacer le nouveau dictionnaire, à l'aide de ces boutons, exactement à l'endroit de la liste où se situait celui qu'il doit remplacer.

Voyez aussi les options relatives aux dictionnaires (traitement des doublons, des mots en minuscules ou accentués, et choix des couleurs des mots)

---

retour menu

retour mode d'emploi.

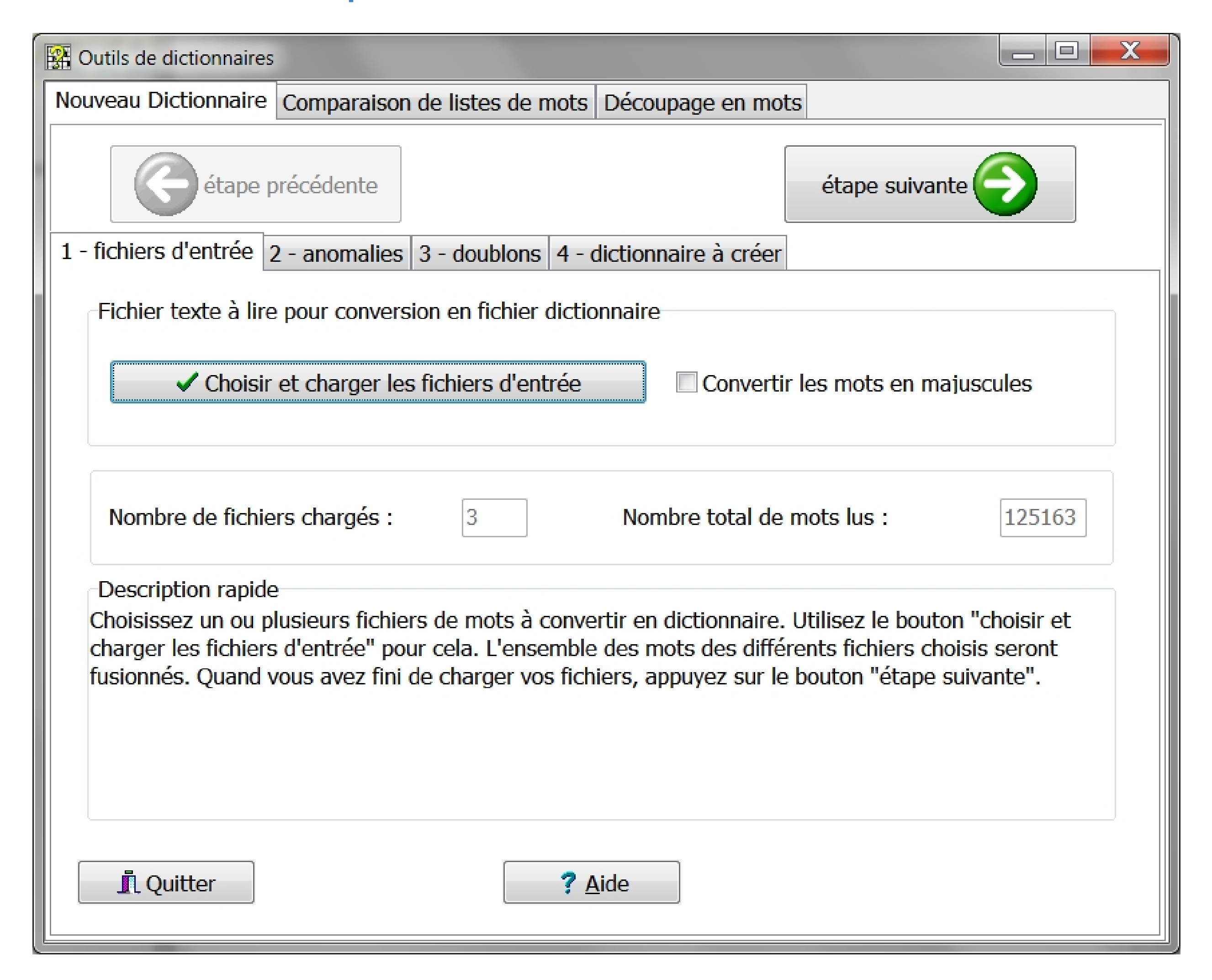
Création de dictionnaire

Création de dictionnaire

Fonctionnement de l'outil accessible par le menu **Outils->Nouveau dictionnaire**

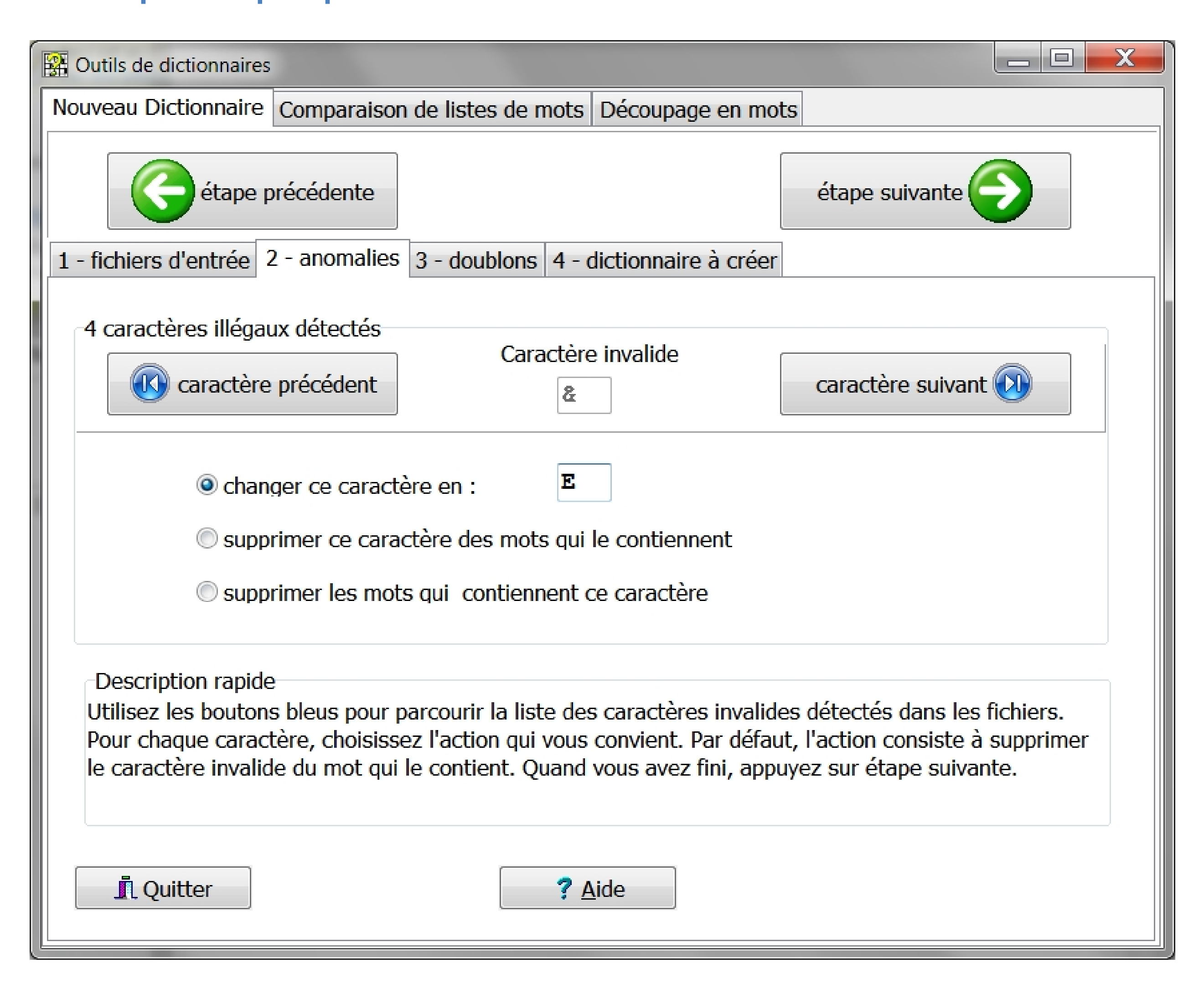
Cet outil vous propose d'organiser la création d'un dictionnaire en 4 étapes successives. La matière première nécessaire est une ou plusieurs listes de mots, un mot par ligne. Il n'est pas nécessaire que ces listes de mots soient triées, ni qu'elles soient épurées des signes non-alphabétiques comme tirets, apostrophes,etc... Si vous disposez d'un texte de plusieurs lignes, AJL vous propose un outil capable d'extraire les mots de ce texte. Voyez l'extracteur de mots. Il peut être un excellent moyen pour alimenter en mots cet outil de création de dictionnaire.

ONGLET 1 - Etape de choix des listes de mots en entrée



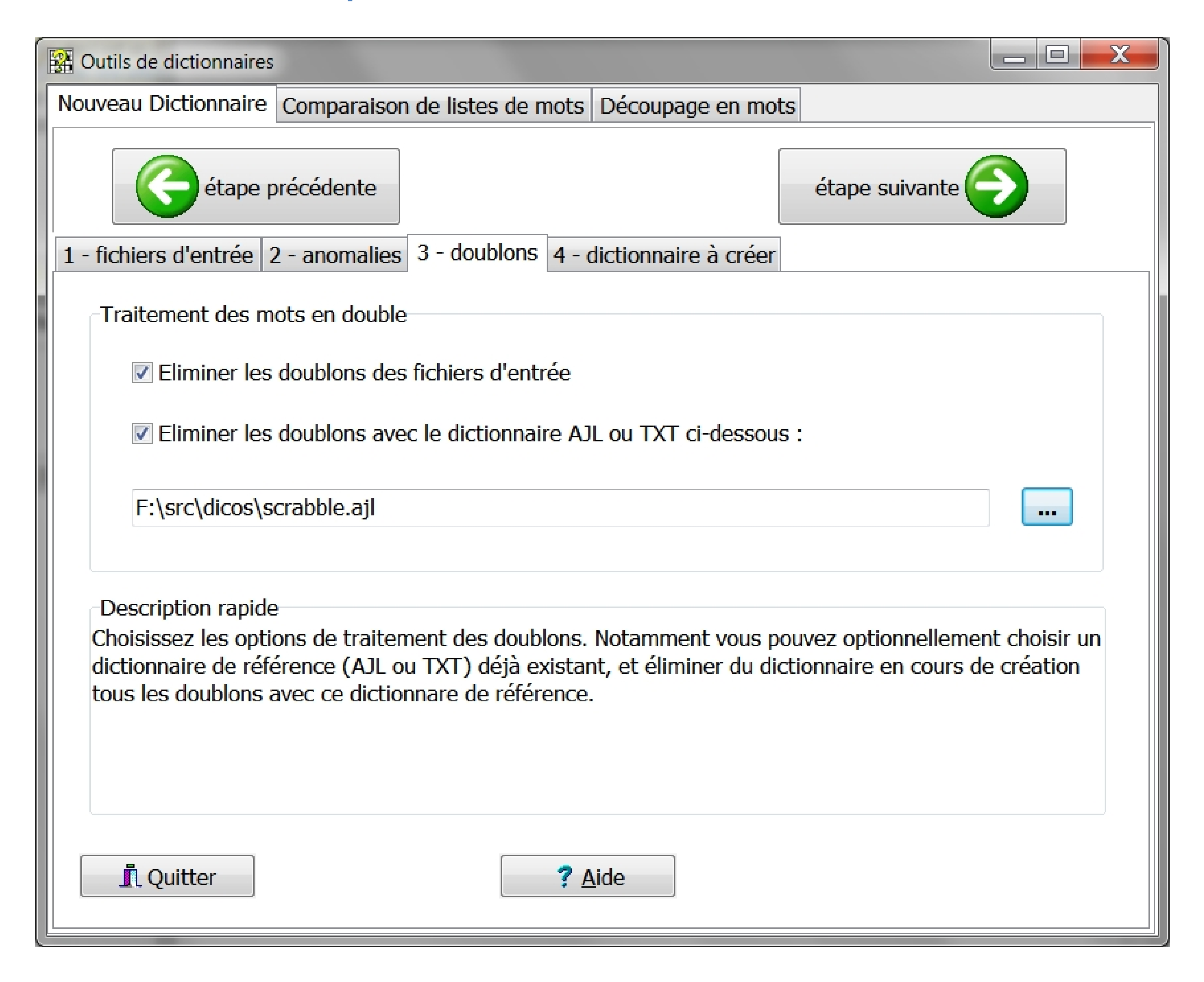
Utilisez le bouton "choisir et charger les fichiers d'entrée" pour choisir les fichiers contenant des listes de mots. Si vous souhaitez rester dans le mode simplifié où AJL traite tout en majuscules, cochez la case prévue à cet effet. Puis appuyez sur "**Etape suivante**".

ONGLET 2 - Etape de traitement des éventuels caractères non-alphabétiques présents dans vos listes



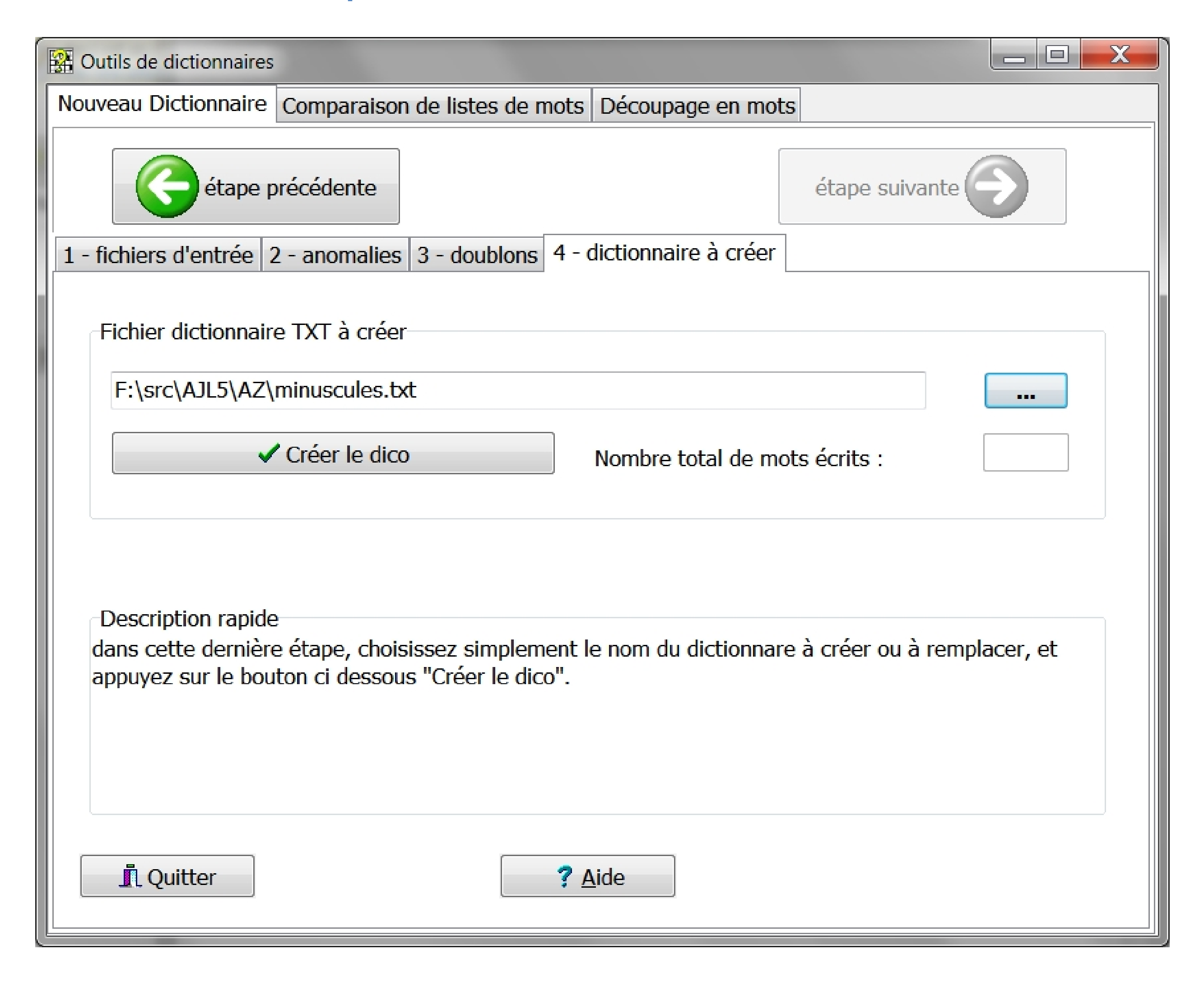
Cet onglet ne s'affiche que si des caractères non assimilables à des lettres sont rencontrés : des espaces blancs, tirets, apostrophes, etc... Vous devez choisir le sort que vous leur réserverez : remplacement, suppression, ou carrément suppression du mot. Traitez toutes les anomalies en navigant avant/arrière dans leur liste, grâce aux deux boutons bleus. Puis appuyez sur "**Etape suivante**".

ONGLET 3 - Etape de traitement des mots en double



Cet onglet vous permet de choisir le sort réservé aux doublons. Il y a deux sources possibles de doublons :
1) ceux déjà en double dans vos fichiers d'entrée. La case à cocher prévue à cet effet permet de les traiter.
2) ceux qui doublonnent avec un dictionnaire déjà créé. Ce peut être un dictionnaire standard AJL comme dans l'exemple ci-dessus, ou un dictionnaire "à vous" au format TXT.
Puis appuyez sur "**Etape suivante**".

ONGLET 4 - Etape de création du dictionnaire de sortie

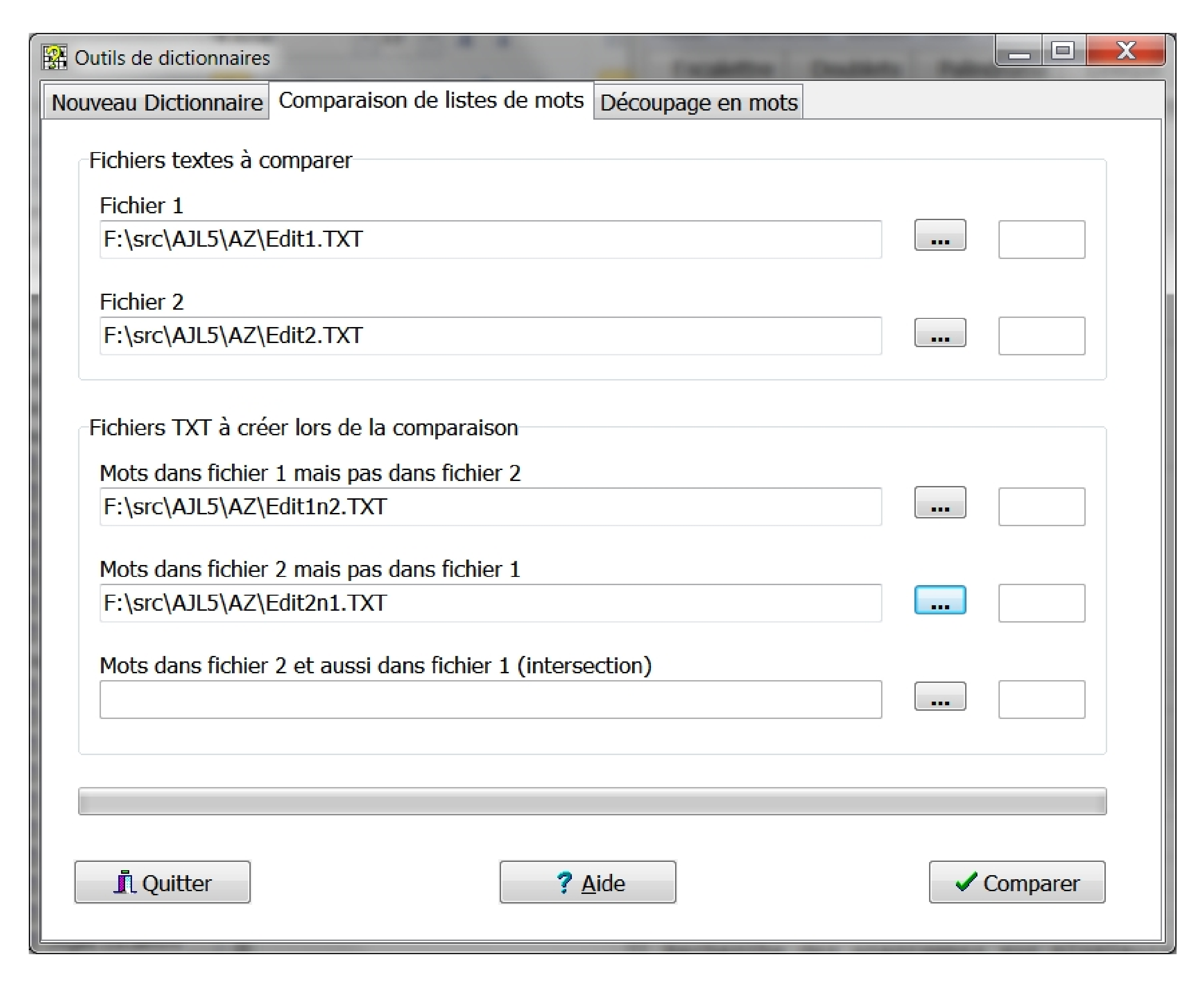


Le travail est achevé ici, avec le choix d'un fichier de sortie qui contiendra votre dictionnaire flambant neuf. Choisissez un nom et un répertoire de stockage avec le bouton **"..."**. Attention, si vous choisissez un fichier existant il sera impitoyablement écrasé sans préavis ! Quand vous êtes prêts, appuyez sur "**Créer le dico**".

Comparaison de fichiers

Comparaison de listes de mots

Fonctionnement de la boîte de dialogue accessible par le menu **Outils->Comparaison de fichiers...**



Cet outil compare entre elles deux listes de mots en format texte. Il n'est pas nécessaire que ces listes soient triées, il suffit qu'elles ne contiennent qu'un mot par ligne. Vous commencez par choisir les deux listes repérées "Fichier1 et "Fichier2" dans l'outil. Pour celà, utilisez les boutons sobrement intitulés **"..."**.

Puis vous pouvez choisir de générer tout ou partie, à votre convenance, des trois fichiers de sortie suivants :

- "mots dans le fichier1 mais pas dans le fichier2"
- "mots dans le fichier2 mais pas dans le fichier1"
- "mots dans le fichier1 et également dans le fichier2"

Appuyez sur le bouton "Comparer", quand vous avez fait vos choix toujours avec les boutons **"..."**.
AJL affiche alors le nombre de mots correspondant à chaque fichier, que vous pouvez examiner avec votre éditeur de texte favori.

Extraction de mots

Extraction de mots

Fonctionnement de la boîte de dialogue accessible par le menu **Outils->Extraire les mots**

Cet outil n'est pas réalisé en natif AJL, mais réalisé à l'aide d'un script en langage EEA, c'est tout de même plus commode...

La matière d'entrée peut être n'importe quel texte en n'importe quelle langue, du moment que le texte soit enregistré sous forme de fichier texte ".txt" simple. Deux boutons vous sont proposés, selon que vous souhaitez ou non convertir au passage en majuscules tous les mots extraits.

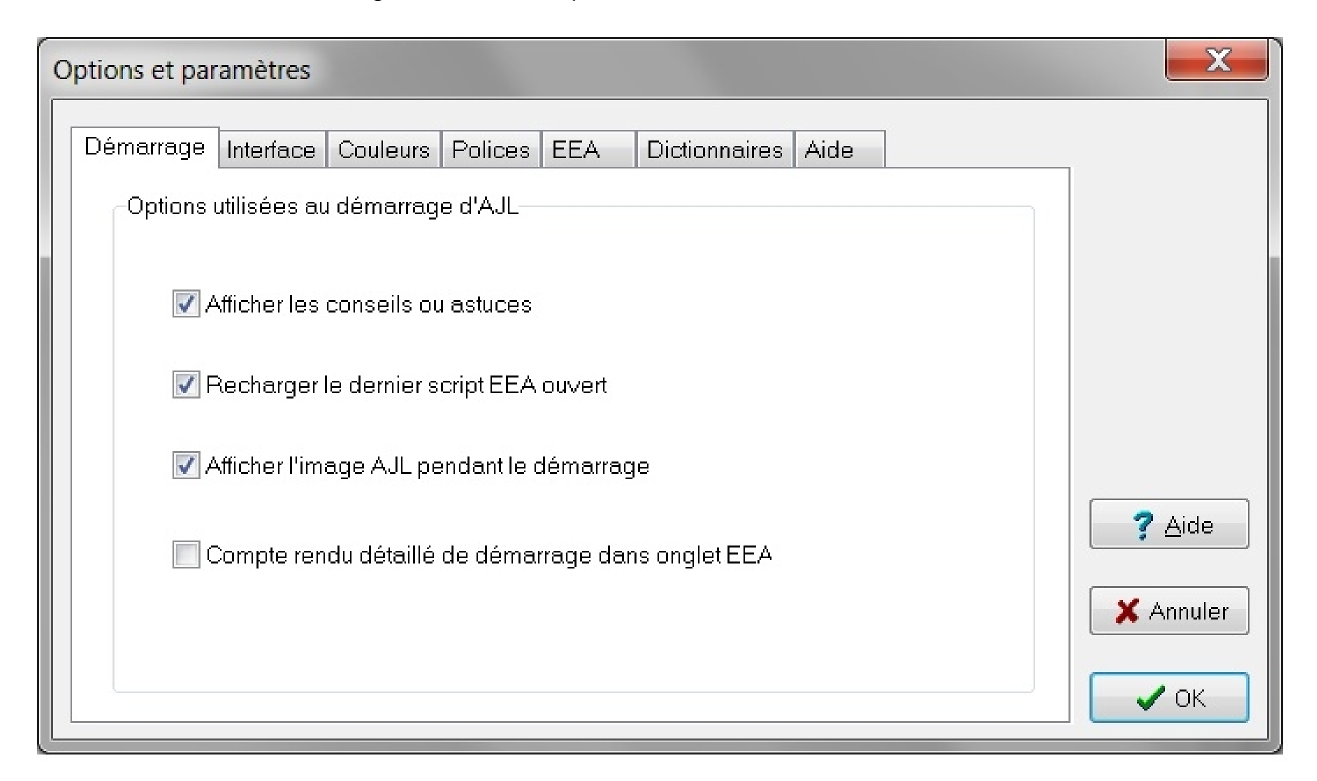
Quand vous avez appuyé sur le bouton choisi, vous vous retrouvez dans l'onglet EEA, où il vous suffit de faire "OK" et de vous laisser guider. Le script vous demandera de choisir un fichier d'entrée, celui qui contient votre texte, et un fichier de sortie, celui qui contiendra la liste des mots extraits. Pour le second fichier, si vous faites "Annuler" dans le dialogue Windows de choix du fichier, le script EEA affichera simplement la liste des mots extraits. Vous devez avoir installé AJL correctement pour bénéficier de ce script, notamment vous devez avoir extrait tous les fichiers du zip dans le même répertoire.

Pour les informaticiens, l'analyse du script EEA en question est un bon exercice qui met en valeur certaines caractéristiques du langage. Pour les autres, il suffit de se laisser guider.

Options générales

Paramètres -> Options générales

On obtient une fenêtre à 7 onglets, décrits ci après.



Démarrage

Affiche les conseils ou astuces
Si ce choix est coché, AJL affiche une astuce ou un conseil différent à chaque démarrage du logiciel.

Recharger le dernier script EEA ouvert.
Avec ce choix coché, et si vous avez fermé AJL alors que l'onglet EEA était l'onglet en cours (et seulement dans ce cas), le dernier script contenu dans cet onglet sera automatiquement chargé lors du prochain démarrage de AJL.

Affiche l'image AJL pendant le démarrage

Si ce choix est coché, une (superbe, merci JC) image de synthèse représentant les 3 jetons de Scrabble "A", "J", et "L" sont affichés pendant que les dictionnaires sont chargés en mémoire. L'image disparaît dès que cette opération est terminée, de sorte de l'affichage de l'image n'occasionne pas de ralentissement du démarrage d'AJL.

Compte rendu détaillé de démarrage dans onglet EEA
Sans réel objet, c'est plutôt une option destinée à mon propre usage en temps de débugging.

Interface

Interface mode débutant
AJL affiche seulement 5 onglets (Inclus, Masque, Contient, Ne Contient Pas, Anagramme), et simplifie certaines options disponibles. Cette option est recommandée aux nouveaux venus, car elle permet de se familiariser sans difficultés avec AJL.

Interface mode expert
AJL affiche tous les onglets disponibles, et active toutes les possibilités "avancées". Cette option est recommandée dès que vous maîtrisez les fonctionnalités proposées dans l'option "interface mode débutant" .

Couleurs

Il est possible de colorer les mots selon leur dictionnaire de provenance. ceci est très utile si, quand vous utilisez plusieurs dictionnaires, vous souhaitez facilement privilégier les mots de certains d'entre eux dans vos recherches (exemple : création de grille de mots croisés à thème). Pour activer cette option, il faut :

1. cocher la case "Colorer les mots"
2. sélectionner un dictionnaire dans la liste affichée. Il suffit de cliquer sur le dictionnaire choisi.
3. appuyer sur le bouton "Couleur..." et cliquer sur la couleur choisie pour les mots de ce dictionnaire.

Dès lors, le nom du dictionnaire est coloré selon votre choix, ce qui vous permet de juger immédiatement de l'effet esthétique qui sera obtenu dans la liste de mots. Procédez de même pour tous les dictionnaires (par défaut, la couleur est noire). Ces choix sont mémorisés pour les prochains démarrage d'AJL.

Remarquez que si vous désirez préserver l'association entre couleurs et dictionnaires lors d'un remplacement d'un des dictionnaire de la liste, des boutons destinés à cet effet sont à votre disposition dans la fenêtre de gestion des dictionnaires.

Polices

Police des zones d'affichage
Choix de la police de caractères utilisée pour l'affichage des mots à l'écran. Il est conseillé de choisir une police non-proportionnelle (exemple : Courrier) afin que des mots de même nombre de lettres aient visuellement la même longueur. Personnellement, j'aime assez la "Fixedsys" de taille 11 ou la "Courrier new" de taille 10. Pour vous permettre de juger facilement de l'effet produit, la boite de dialogue de choix d'une police contient un bouton "Appliquer" qui applique immédiatement la police choisie à la zone d'affichage des listes de mots.
Police générale d'écran
Choix de la police de caractères utilisée pour l'affichage des libellés fixes des écrans : nom des champs, des onglets, des textes d'aide, etc.. Personnellement, j'aime assez la "Garamond" de taille 10. Pour vous permettre de juger facilement de l'effet produit, la boite de dialogue de choix d'une police contient un bouton "Appliquer" qui applique immédiatement la police choisie à l'onglet Options lui même.
Vous disposez d'une case à cocher pour appliquer ce choix aux élements de menu, bien que ceux ci soient

usuellement directement pilotés par Windows et normalement non modifiables.
Police d'imprimante
Choix de la police de caractères utilisée pour l'impression de la liste des mots.

Tous ces choix sont mémorisés et reproduits à chaque démarrage.

EEA

Effacer la zone message à chaque exécution
La zone message, qui est la zone d'affichage cible des instructions MESSAGE ainsi que de différents messages d'information automatique, est dimensionnée pour contenir 1000 lignes. Cochez ce choix afin de vider cette zone à chaque exécution de script (bouton OK). Les messages les plus récents remplacent les plus anciens si le nombre maximal de 1000 lignes est atteint.

Afficher le contenu des variables à chaque interruption
Si cette option est cochée, et qu'un script EEA est interrompu lors de son exécution, par exemple par l'action du bouton "
Interrompre", ou par une instruction PAUSE, le contenu de toutes les variables connues du script EEA est affiché dans la zone message par une ligne du type : [Debug] nom = valeur. Ceci peut s'avérer utile pour tester le bon déroulement d'un script long et complexe.

Afficher le contenu des variables en cas d' erreur
Identique au cas précédent, l'événement déclencheur de l'affichage des variables étant cette fois une erreur rencontrée lors de l'exécution du script. Exemples : utilisation de variable inconnues, division par zéro, etc... Il est conseillé de laisser cette option cochée, afin de faciliter la correction de vos erreurs de script.

Interrompre à chaque 1000 ème sélection
La zone liste de mots, qui est la zone d'affichage cible des instructions SELECTION est dimensionnée pour contenir 1000 lignes. Cochez ce choix pour interrompre automatiquement le script EEA lorsque cette zone est remplie. Vous pourrez alors examiner ou sauvegarder les 1000 lignes affichées, et poursuivre l'exécution du script (bouton Poursuivre). Sinon, cette zone sera vidée automatiquement dès qu'une 1001 ème ligne doit y être affichée.

Paramètres système
Ces paramètres régissent la gestion mémoire et le multi-threading. N'y touchez sous aucun prétexte sous peine de dégrader sévèrement les performances de AJL. Si vous programmez en expert des fonctions EEA hautement récursives ou multi-thread, allez lire ce paragraphe.

Dictionnaires

Elimine les mots en double
Si ce choix est coché, AJL ne retient qu'un seul exemplaire d'un mot qui se trouverait présent dans plusieurs dictionnaires.

Affiche le nombre de mots des dictionnaires chargés
Si ce choix est coché, AJL indique au démarrage du logiciel, ainsi qu'à chaque changement de dictionnaire, le nombre de mots et la liste des dictionnaires utilisés.

Aide

Choisissez parmi les sites proposés en liste déroulante celui que vous voulez utiliser pour afficher les définitions des mots. En effet, depuis n'importe quel onglet AJL, si un mot de la liste de mots affichés (cadre de droite) est sélectionné et que le bouton Aide est activé, AJL complète automatiquement l'URL choisi dans cette liste par la version en lettres minuscules du mot choisi, et appelle votre navigateur Internet par défaut.

Vous pouvez aussi saisir un site de votre choix en lieu et place d'un de ceux proposés. Veillez bien à ce que le site accepte d'être interrogé simplement en complétant son URL par le mot cherché. Par exemple `"http://www.definition-des-mots.fr/choixdumot/`**`mot`**`"`

Dans ce cas, vous taperez simplement "`http://www.definition-des-mots.fr/choixdumot/`" dans la zone de saisie.

---

retour menu

retour mode d'emploi.

Valeur des lettres

Paramètres -> valeur des lettres

L'entrée de menu **Paramètres->valeur des lettres** affiche une boîte de dialogue qui permet d'assigner une valeur numérique à chaque lettre. Cette association permet donc de calculer la "valeur" d'un mot, en additionnant la valeur de chacune de ses lettres. Ceci est utile dans l'onglet Expert, qui dispose des fonctions valeur min, et valeur max, dans le tri, qui permet de trier la liste des mots selon cette valeur, et dans les scripts EEA où la fonction Valeur permet de calculer la valeur d'un mot en fonction de la valeur des lettres.

Dans le langage EEA, la fonction Poids permet d'effectuer dans un script la même association lettre<->valeur que la boîte de dialogue "Valeur des lettres" permet de faire.



Fonctionnement de la boîte de dialogue accessible par le menu Paramètres->Valeur des lettres

1) Boutons à effet global

- Les grandes flèches sur la droite de la boîte incrémentent ou décrémentent la valeur de toutes les lettres
- Le bouton "Scrabble" attribue aux lettres leur valeur dans le Scrabble (Français).
- Le bouton "Zéro" remet à zéro la valeur de toutes les lettres
- Le bouton "Rang" affecte à chaque lettre une valeur égale à son rang (1 à 26) dans l'alphabet.

2) Raccourcis claviers:

- Si vous appuyez sur une lettre au clavier, sa valeur augmente de 1.
- Si en même temps, vous appuyez sur SHIFT, sa valeur diminue de 1.
- Si au lieu de SHIFT vous appuyez sur CTRL, sa valeur est mise à 0.

3) Réglage fin des 26 compteurs

- Vous pouvez aussi taper directement la valeur souhaitée dans le champ prévu à cet effet dans chacun des 26 compteurs, ou utiliser les petites flèches haut/bas de chaque lettre pour modifier la valeur.
- Les touches "flèches en haut" et "flèche en bas" du clavier agissent aussi sur le compteur sélectionné.

Vos choix seront conservés au prochain démarrage d'AJL.

---

retour menu

retour mode d'emploi.

Symboles phonétiques

Paramètres -> Symboles phonétiques

AJL, permet de traiter un autre type de dictionnaire, dit phonétique. Les mots n'y sont plus orthographiés de manière conventionnelle, mais représentés selon une suite de phonèmes, chaque phonème possédant son propre symbole, normalisé par l'**API (Association phonétique internationale**). Malheureusement, cette association a cru bon d'utiliser pour symboles un jeu de caractères nécessitant une police très particulière pour être affichés ou imprimés, et de toutes façons, aucun clavier standard ne les représente.

AJL utilise donc son propre jeu de symboles, qui a été choisi de façon arbitraire mais le plus logiquement possible. Comme tout le monde a le droit de ne pas partager mes choix, la fenêtre "Choix des symboles phonétiques" permet de faire vos propres associations entre phonèmes et symboles.

Norme API	Norme AJL	Exemple	Norme API	Norme AJL	Exemple
p	p	pas, cap	i	i	il, habit, dîne
t	t	tu, étale, lutte	e	é	thé, fermer, blé
k	k	caste, que, képi, accueil	ɛ	è	dais, être, procès
b	b	bien, abîme, club	a	a	crabe, papa, nappe, tache
d	d	don, broder, bled	ɑ	A	âne, pâle, tâche, mât
g	g	gare, vague, zigzag	ɔ	o	or, robe, donne,
f	f	fou, affreux, chef	o	O	dos, chevaux, rose
v	v	vite, ouvrir	u	U	doux, genou, roue
s	s	souffle, chasse, hélas	y	u	user, tu, sûr
z	z	rose, zébu, maison	œ	ê	cœur, peur, neuf
ʃ	H	choix, tache, auch	ø	E	deux, vœu, peu,œufs
ʒ	j	âge, jus, geôle, page	ə	e	le, monsieur, demain
l	l	large, molesse, mal	ɛ̃	î	plein, lin, pain, sein
r	r	rude, mari, ouvrir	œ̃	û	alun, parfum
m	m	maison, amener, blême	ɑ̃	â	vent, sans, paon, temps
ɲ	N	ignoble,digne,bagne	ɔ̃	ô	ondée, bon, honte
n	n	nourrir, fanal, dolmen	j	ï	lieu, yeux
x	x	espagnol de jota	ɥ	ÿ	huile, lui
ŋ	G	planning, ring	w	ü	oui, louis

De plus, comme certains phonèmes distincts sont très voisins (exemple le "in" de lin et le "un" de lundi), trois boutons vous proposent un regroupement des sons les plus voisins:

Bouton FIN

si vous voulez suivre strictement la norme API. C'est le choix par défaut, il identifie 12 voyelles simples, 4 voyelles nasales, 3 semi-voyelles soit 19 vocales. Elles sont complétées de 20 consonnes, si l'on tient compte du "h" aspiré de non-liaison.

Bouton MOYEN :

- regroupe le "E" de deux, voeu, peu avec le "e" de le, demain, premier.
- regroupe le "un" de alun parfum, lundi avec le "in" de pain, sein, lin
- regroupe le "r guttural" espagnol de jota avec le "r" usuel.

- regroupe le "A" de âne, pâle, mât avec le "a" de crabe, papa, nappe

Bouton GROSSIER

Même effet que le bouton MOYEN, mais en plus :

- regroupe le "eu" de coeur, peur, neuf avec le "e de le, demain, premier
- regroupe le "O" de dox, chevaux, rose avec le "o" de or, robe, donne
- regroupe le "u" de huile, lui, nuit avec le "u" de tu, user, sûr
- regroupe le "ou" de loin, ouest, oui avec le "ou" de doux, genou, roue.

Il est bien sûr possible de retoucher ces regroupements comme bon vous semble. Vos choix seront conservés au prochain démarrage d'AJL.
Remarque : dans la version 2 du dictionnaire phonetique, les phonèmes h aspiré (noté ') et A ne sont pas identifiés.

---

retour menu

retour mode d'emploi.

Le monde EEA

le monde EEA

EEA signifie Evaluateur d'Expressions AJL.

Qu'est ce que c'est ?
- un langage informatique très simplifié
- un interpréteur, c'est à dire un programme capable de comprendre et d'exécuter une suite d'instructions écrites en langage EEA
- une fenêtre de travail, constituée par l'onglet EEA, qui permet la rédaction, le test, l'exécution et la sauvegarde de programmes écrits en EEA.

Tenté par un essai ?
Alors, essayez l'un des nombreux scripts EEA prêts à l'emploi : Menu Fichier-> Ouvrir script EEA. Choisissez au hasard un des scripts parmi ceux installés automatiquement avec AJL, cliquez sur OK, c'est tout !

A quoi ca sert, EEA ?
AJL dispose de très nombreuses fonctions, néanmoins il y a parmi vous d'insatiables experts (allez, les Oulipiens, dénoncez-vous!) qui veulent aller toujours plus loin qu'AJL et ses fonctions toutes faites. Maintenant, avec EEA, AJL est un produit totalement ouvert, et si vous avez besoin d'une fonction particulière, il vous suffit de n'avoir pas peur de faire un peu d'informatique pour créer vos propres outils ! A vous les nuits blanches, à moi la paix !!

Eléments du langage EEA

Onglet EEA et fenêtres de travail

Bibliothèque d'exemples.

Astuce relative à l'appel à l'aide en ligne dans AJL :

Depuis l'onglet EEA du programme AJL, pour obtenir directement l'aide en ligne relative à une instruction ou une fonction, mettez simplement en surbrillance l'élément sur lequel vous voulez de l'aide, et appuyez sur le **bouton Aide**.

Accès direct à la documentation de référence du langage EEA
Les 6 instructions d'affectation,

Liste des opérateurs et fonctions

Liste des instructions de bloc

Bibliothèque d'exemples
Intimidé par EEA ?
Alors inspirez vous de la bibliothèque d'exemples. Vous pouvez aussi utiliser le menu Fichier -> Ouvrir script EEA et lancer un des nombreux scripts prêts à l'emploi livrés dans l'installation de AJL.

Que faire en cas de plantage ?
Malgré tout le soin apporté au développement de EEA, il peut arriver que des erreurs m'aient échappé lors des tests. Il se peut qu'un script EEA s'arrête anormalement avec ce message d'erreur **`Erreur interne à l'exécution de la ligne x`**, voir même, pire, qu'une erreur Windows soit générée. Dans ce cas, prenez contact avec moi et je m'efforcerai de corriger au plus vite l'anomalie en question.

retour Mode d'emploi

retour Introduction

Eléments du langage EEA

Syntaxe générale du langage EEA

Le langage EEA est assez rudimentaire, et nul besoin de faire appel au formalisme habituel des manuels informatiques pour le décrire. Il suffit de s'entendre sur quelques éléments de vocabulaire. Commençons par dire qu'un programme EEA est une suite de lignes, chacune appelée instruction.

instruction

c'est une ligne dont la syntaxe correspond à une des instructions décrites dans cette aide, ou un commentaire. Un commentaire est un texte libre commençant par deux barres obliques //. Une ligne peut comporter à la fois une instruction et un commentaire, selon la syntaxe :

```
instruction  //commentaire
```

constante

C'est un nombre entier (exemple : 456), décimal (exemple : 3.1416) ou une chaîne de caractères que l'on encadre par des guillemets (exemple : "AZERTY").

Le caractère d'échappement '\' permet la définition d'une chaîne de caractères comportant un guillemet. Exemple `"c'est \"cool\" cet AJL"` donne la chaîne `c'est "cool" cet AJL`
Pour inclure le caractère d'échappement lui même dans une chaîne, par exemple un nom de fichier, il suffit de le doubler : `"c:\\ajl\\ajl.exe"`

Un autre usage de ce caractère est de permettre l'écriture d'une instruction sur plusieurs lignes. Normalement, une instruction occupe une seule ligne, mais certaines instructions peuvent exiger des zones de texte très longues, comme par exemple les instructions destinées à afficher un message. Dans ce cas, on peut interrompre une instruction en terminant sa 1ère ligne par le caractère \, et la poursuivre immédiatement en début de la ligne suivante. Exemple :

```
Message("Ceci est une très \
longue instruction \
qui s'étale sur \
quatre lignes")
```

variables, tableaux, listes

Une variable est un conteneur permettant le stockage de nombres (tout comme la touche "M" des calculatrices) ou de chaînes de caractères. Vous pouvez en utiliser autant que vous voulez. Une variable est identifiée par son nom. Il est également possible d'utiliser des éléments de tableaux comme variable. EEA supporte les tableaux multi-dimensionnels et les indices de tous les types. Enfin, EEA propose une structure de type liste.
Plus de détails ici sur les variables, tableaux, listes.

expression

C'est une combinaison légale (c'est à dire que chaque opérateur ou fonction "trouve" bien ses arguments) de variables, opérateurs, fonctions, constantes entières ou chaînes de caractères, agrémentée éventuellement de parenthèses. Exemple intuitif : 2 * 3 est une expression utilisant les valeurs 2 et 3 et l'opérateur multiplication *, ce qui donne une nouvelle valeur, en l'occurrence 6.
Si nécessaire, on peut utiliser des parenthèses pour faire des regroupements.
Exemple : (1 + 1) * (5 - 2) utilise l'opérateur + dans 1+1, qui donne 2, l'opérateur - dans 5-2 qui donne 3, l'opérateur * avec les résultats précédents, soit 2 * 3 qui donne 6.

opérateur

Il agit sur deux expressions, selon la syntaxe `expression1 opérateur expression2`, comme dans 2 * 3., ou (1+1)*(5-2). Le résultat de l'action d'un opérateur est une autre expression, de type entier, ou chaîne de caractère, ou booléen Vrai / Faux dans le cas d'opérateurs de comparaison. Les éléments sur lesquels agit un opérateur peuvent être des expressions complexes.
Voyez la liste des opérateurs ici.

fonction

Elle agit sur une ou plusieurs expressions dites arguments de la fonction, entre parenthèses et séparés par des virgules. Par exemple `Fonction(expression1,expression2)` pour une fonction à 2 arguments. Son résultat est une autre expression, de type entier, ou chaîne de caractère, ou les "booléens" VRAI ou FAUX . Par exemple la fonction Longueur appliquée à la constante "AZERTY" : Longueur ("AZERTY") donne le résultat entier 6. Une fonction peut aussi ne renvoyer aucun résultat, par exemple la fonction Message("texte"). Les fonctions sans valeur de retour forment à elles seules une instruction valide, et doivent donc être seule sur leur ligne. Les fonctions renvoyant un résultat doivent en revanche être employées au sein d'une expression utilisant le résultat rendu.
Voyez la liste des fonctions.
Vous pouvez aussi créer vos propres fonctions.

affectation

C'est l'instruction de base, qui permet de calculer une expression, et d'en conserver le résultat dans une variable. Sa syntaxe est `variable = expression`. Exemple : toto = 1 + 1 qui affecte la

valeur 2 à la variable de nom toto. Il existe plusieurs variantes de l' instruction d' affectation, qui permettent par exemple de combiner une addition et une affectation dans une seule instruction, ou d' affecter d'un coup une liste de valeurs à une liste de variables.
Plus de détails ici sur les 6 types d'affectation possibles.

autres instructions

Ce sont les "instructions de bloc" qui permettent de gérer le déroulement logique du programme. Certaines de ces instructions ont des arguments un peu à la manière des fonctions, avec cependant quelques différences :

- les arguments sont séparés par des points virgules,
- les instructions de bloc ne renvoient jamais de valeur.
- Les instructions de bloc peuvent s'écrire indifféremment en minuscules (tanque, si, ...) ou en majuscules (TANTQUE, SI,...),
- La plupart ont un équivalent anglophone inspiré directement du langage de programmation "C" : if, else, while, for,…

Liste des instructions de bloc.

Variables, tableaux, listes

Les variables

Une variable EEA est identifiée par un nom comportant des caractères majuscules, minuscules, des chiffres, ou le caractère '_'. Seule restriction : le nom ne peut pas commencer par un nombre.
Si vous êtes utilisateur EEA expérimenté et que vous définissez vos propres fonctions, lisez aussi le paragraphe décrivant les variables globales.

Exemples:

```
truc12 = 1                  //truc12 est un nom valide
_Bidule_Chouette = 1        //_Bidule_Chouette aussi
1Bidule_pas_chouette = 1    //commence par un nombre: illégal
UnBidule_pas_chouette = 1   //valide.
```

Les types

Les variables EEA peuvent être de 3 types: nombre entier, nombre avec décimale, chaîne de caractères. (Remarque, les booléens VRAI et FAUX sont considérés comme des entiers, respectivement 1 et 0). Le type hexadécimal est une variante du type entier. Pour utiliser une constante hexadécimale, préfixez la valeur de # et utilisez les majuscules pour les digits A à F. L'affichage d'une valeur hexadécimale suit la même convention. Par exemple :
a = #FF.

Le type d'une variable n'est pas prédéterminé et déclaré avant son utilisation, comme dans de nombreux langages, mais variable et déterminé lors de l'exécution. Ainsi, la suite d'instructions suivante est permise:

```
a = 123       //a est nombre entier
```

a = 123.456 //puis un nombre à décimale flottante
a = "azerty"   //et maintenant, une chaîne de caractère

Les tableaux

EEA permet l'utilisation de tableaux à une ou plusieurs dimensions. Un élément de tableau, c'est à dire un nom de tableau complété d'un ou plusieurs indices entres crochets, n'est jamais qu'un nom de variable un peu plus compliqué... Comme pour les variables simples, aucune déclaration préalable de type ou de taille n'est nécessaire:

- tableaux à une dimension :

a = table[0]   //l'indice 0 est permis
table[-8] = a   //même les indices négatifs!
c = table["A"]   //et aussi les indices chaînes de caractères

- tableaux à plusieurs dimensions : tout est permis.

a = table[0][1]
table[-8][0]["b"] = x + x
c = table["AZ"]["ER"]["TY"]

Les listes

Qu'est ce qu'une liste ? C'est une série ordonnée d'objets, rangés dans une sorte de boîte représentée par une paire d'accolades.

De manière très intuitive :

`{07,"mars",1936,"Georges","Perec"}` est une liste de 5 éléments, comportant des nombres et des chaînes de caractères.

`{"Tigre","Lion","Panthère","Escargot"}` est une autre liste.

EEA permet dispose d'outils permettant d'explorer les éléments d'une liste et de nombreuses fonctions spécifiques aux listes. Les listes sont de très puissants outils, et la seule "structure" de donnée que gère EEA.

Un article, toutefois assez technique, en donne une présentation complète.

retour Syntaxe générale EEA

Affectations

Les affectations

Affectation simple « = »

Cette instruction permet de calculer une expression, et d'en conserver le résultat dans une variable.

Syntaxe : variable = expression

Exemples :
`Toto = 1 + 1`
affecte la valeur 2 à la variable Toto

`toto = DEBUT("MAMAN",2)   +  FIN("BASQUE",4)`
affecte la valeur "MA" + "SQUE", soit "MASQUE", à la variable toto

Attention, ne confondez pas avec l'opérateur ==, qui dans : variable == expression teste si la variable a la même valeur que l'expression, et dont le résultat est le booléen OUI ou NON (un ou zéro) .

L' incrément « += »

Cette instruction permet de calculer une expression, et d'additionner le résultat au contenu d'une variable.

Syntaxe : variable += expression
Il s'agit d'une forme strictement équivalente à : `variable = variable + expression`
Son intérêt réside dans sa compacité d'écriture, et aussi dans le fait que son exécution réclame moins de temps que la forme utilisant séparément "=" et l' opérateur d' addition « + »

Exemple
`i += 1` // ajoute la valeur 1 au contenu de la variable i.

Le décrément « -= »

Cette instruction permet de calculer une expression, et de retrancher le résultat au contenu d'une variable.

Syntaxe : variable -= expression
Il s'agit d'une forme strictement équivalente à : `variable = variable - expression`
Son intérêt réside dans sa compacité d'écriture, et aussi dans le fait que son exécution réclame moins de temps que la forme utilisant séparément "=" et l' opérateur d' addition « - »

Exemple
`j -= 1` // retire la valeur 1 au contenu de la variable j.

Affectation multiple « = »

On peut aussi affecter une suite de valeurs à une suite de variables. Les deux suites doivent être de même taille.

Syntaxe variable1, variable2, …, variableN = valeur1, valeur2, …, valeurN
Il s' agit d' une forme strictement équivalente à la suite d' affectations simple suivante :
variable1 = valeur1
variable 2 = valeur2
… etc…
variable N = valeurN

Exemples
`Mot , longueur =  "A" , 1`   // affecte "A" à Mot et 1 à longueur

```
a , b = b , a
```
//manière élégante d'échanger deux valeurs en une seule instruction !

Incrément et décrément multiple « += » et « -= »

De même, on peut réaliser des incréments multiples :

variable1, variable2, …, variableN += valeur1, valeur2, …, valeurN
strictement équivalent à
variable1 += valeur1
variable 2 += valeur2
… etc…
variable N += valeurN

et des décréments multiples
variable1, variable2, …, variableN –= valeur1, valeur2, …, valeurN
strictement équivalent à
variable1 -= valeur1
variable 2 -= valeur2
… etc…
variable N -= valeurN

retour Syntaxe générale EEA

Abrégé de grammaire

La grammaire de EEA

Nota : cet article s'adresse aux personnes recherchant une approche du langage plus rigoureuse et moins intuitive que celle développée dans les deux articles précédents. La lecture de cet article n'est en aucun cas indispensable pour utiliser le langage EEA.

En théorie de l'informatique, la grammaire d'un langage décrit les règles formelles que doit respecter toute liste d'instructions écrite dans ce langage. EEA étant un langage très simple, on peut le décrire de manière suffisamment précise avec seulement 2 entités abstraites, décrivant pour l'une un contenu, pour l'autre un contenant :

1. **une valeur droite.** C'est une expression qui produit un résultat, typiquement ce qu'on trouve à droite d'un signe égal dans une instruction d'affectation. On notera cette entité VALD. Par exemple, "5" ou "(Longueur(a)+2)*3" sont des VALD. L'exemple précédent montre que les VALD peuvent se combiner entre elles, à l'aide d'opérateurs et de fonctions, pour former une autre VALD. Plus précisément :
    - VALD <-- *Identifiant*
    - VALD <-- *Identifiant*[VALD]...[VALD] (tableau à indices multiples)
    - VALD <-- {VALD, ...VALD} (listes)
    - VALD <-- (VALD) (parenthésage d'une expression)
    - VALD <-- *Nomfonction*(VALD, ... ,VALD) (fonction de plusieurs arguments)

- VALD <-- VALD *Operateur* VALD (opérateur binaire)
- VALD <-- *Operateur* VALD (opérateur unaire avec argument à droite)
- VALD <-- VALD *Operateur* (opérateur unaire avec argument à gauche)
- VALD<-- *VALD (pointeur)

2. **une valeur gauche.** C'est un contenant pour une VALD, typiquement ce qu'on trouve à gauche d'un signe égal dans une instruction d'affectation. On notera cette entité VALG. Par exemple, "a" ou "Table[2]" sont des VALG. L'exemple précédent montre qu'une VALG peut être soit un simple identifiant de variable : "I*dentifiant*", soit un élément de tableau.
   - VALG <-- *Identifiant*
   - VALG <-- *Identifiant*[VALD]...[VALD]
   - VALG <--*VALD (pointeur)

3. exceptions : quelques fonctions ou opérateurs utilisent des VALG comme argument:
   - VALD <-- *Existe(VALG,VALD)*
   - VALD <-- *TriAlpha(VALG,VALD,VALD)*
   - VALD <-- *TriNum(VALG,VALD,VALD)*
   - VALD <-- *Associe(VALG,VALD)*
   - VALD <-- *Dissocie(VALG,VALD)*
   - VALD <-- VALG++
   - VALD <-- ++VALG
   - *Dicoliste(VALG)*

Dès lors, une instruction EEA doit correspondre à une des 5 catégories suivantes

1. **un appel de procédure**

Une instruction peut se réduire à un unique appel de fonction "*Nomfonction*(VALD)" ou "*Nomfonction*(VALD, ... ,VALD)", dès lors que cette fonction ne produit aucune valeur en retour. On appelle ce type de fonction une procédure. Par exemple, l'instruction "Message(a)" est permise, mais "Longueur(a)" est interdite, car la fonction Longueur produit un résultat, qui doit impérativement être utilisé dans une expression.

2. **une affectation**

Une affection doit être construite sur un des 6 modèles possibles suivants :
VALG = VALD
VALG += VALD
VALG -= VALD
formes multiples : autant d'arguments à droite que d'arguments à gauche
VALG, ... ,VALG = VALD, ... ,VALD
VALG, ... ,VALG += VALD, ... ,VALD
VALG, ... ,VALG -=VALD, ... ,VALD

3. **une instruction de début de bloc** *(version anglaise)*

si(VALD) *(if)*
tantque(VALD) *(while)*
boucle(INST;VALD;INST) *(for)*
liredico(VALG)

liredicoex(VALG;VALD;VALD;VALD)
lirefichier(VALD;VALG)
lireliste(VALD;VALG) *(forlist)*
%Nomfonction(VALG, ... ,VALG)

4. **une instruction optionnelle de milieu de bloc** *(version anglaise)*

sinon *(else)*
sinonsi *(elsif)*
arrettantque *(breakwhile)*
suitetantque *(nextwhile)*
arretboucle *(breakfor)*
suiteboucle *(nextfor)*
arretliredico
suiteliredico
arretlirefichier
suitelirefichier
arretlireliste *(breakforlist)*
suitelireliste *(nextforlist)*
exit

5. **une instruction de fin de bloc** *(version anglaise)*

finsi *(endif)*
fintantque *(endwhile)*
finboucle *(endfor)*
finliredico
finlirefichier
finlireliste *(endforlist)*
%retour(VALD, ... ,VALD) *(return)*

retour Syntaxe générale EEA

Instructions opérant en bloc logique

Instructions opérant en blocs logiques

Astuce relative à l'appel à l'aide en ligne dans AJL :
Depuis l'onglet EEA du programme AJL, pour obtenir directement l'aide en ligne relative à une instruction ou une fonction, mettez simplement en surbrillance l'élément sur lequel vous voulez de l'aide, et appuyez sur le **bouton Aide**.

Liste des instructions de début et de fin de bloc, avec les instructions optionnelles d'intérieur de bloc.

Remarques générales pour les instructions de bloc :

1) elles sont **insensibles** à la casse. Vous pouvez écrire LIREDICO, Liredico, liredico, LireDico, etc.. comme bon vous semble.
2) leurs arguments, à la différence des fonctions, sont toujours séparés par des **points-virgules**.

Bloc si - finsi (synonymes : if - endif)

- Sinon (synonyme : else)
- Sinonsi (synonyme : elsif)

Bloc tantque – fintantque (synonymes : while – endwhile)

- Suitetantque (synonyme : nextwhile)
- Arrettantque (synonyme :breakwhile)

Bloc boucle – finboucle (synonymes : for – endfor)

- Suiteboucle (synonyme : nextfor)
- Arretboucle (synonyme : breakfor)

Bloc liredico – finliredico

- Suiteliredico
- Arretliredico
- LiredicoEx – Finliredico

Bloc lirefichier – finlirefichier

- Suitelirefichier
- Arretlirefichier

Bloc lireliste – finlireliste (synonymes : forlist - endforlist)

- Suitelireliste (synonyme : nextforlist)
- Arretlireliste (synonyme : breakforlist)

Bloc %xxxx – %retour (synonyme : %return)

- Exit

Bloc si - finsi

Bloc : si() - finsi

Instructions optionnelle de bloc : `sinon, sinonsi`
Synonymes respectifs : `if(),endif, else, elsif`

Ces instructions permettent l'exécution conditionnelle d'un bloc d'instructions si le résultat d'un test est VRAI.

On peut aussi enchaîner plusieurs tests conditionnant chacun l'exécution d'un bloc d'instructions (instruction SINONSI), et aussi définir un bloc d'instructions qui ne sera exécuté que si aucun des tests précédents n'a abouti à un résultat VRAI (utilisation de l'instruction SINON).

La syntaxe permet donc plusieurs formes. Notez que, s'il existe, le bloc `sinon` doit être obligatoirement en dernière position, et que vous pouvez utiliser autant de blocs `sinonsi` que désiré.

- bloc si - finsi

```
si(expression)
     bloc d'instructions 1
finsi
```

L'expression entre parenthèses dans le SI est évaluée. Si et seulement si elle est VRAIE, le bloc d'instructions 1 est exécuté.

- blocs si - sinon - finsi

```
si(expression)
     bloc d'instructions 1
sinon
     bloc d'instructions 2
finsi
```

L'expression entre parenthèses dans le SI est évaluée. Si elle est VRAIE, seul le bloc d'instructions 1 est exécuté. Si elle est FAUSSE, seul le bloc d'instructions 2 est exécuté.

- blocs si - sinonsi - finsi

```
si(condition 1)
     bloc d'instructions 1
sinonsi (condition2)
     bloc d'instructions 2
sinonsi (condition3)
     bloc d'instructions 3
finsi
```

L'expression entre parenthèses dans le SI est évaluée. Si elle est VRAIE, seul le bloc d'instructions 1 est exécuté. Si elle est FAUSSE, la condition 2 est évaluée. Si elle est VRAIE, seul le bloc d'instructions 2 est exécuté. Si elle est FAUSSE, la condition 3 est évaluée. Si elle est VRAIE, seul le bloc d'instructions 3 est exécuté. Si elle est FAUSSE, aucun bloc d'instructions n'est exécuté.

- blocs si - sinonsi - sinon - finsi

```
si(condition 1)
     bloc d'instructions 1
sinonsi (condition2)
     bloc d'instructions 2
sinonsi (condition3)
     bloc d'instructions 3
sinon
     bloc d'instructions 4
finsi
```

L'expression entre parenthèses dans le SI est évaluée. Si elle est VRAIE, seul le bloc d'instructions 1 est exécuté. Si elle est FAUSSE, la condition 2 est évaluée. Si elle est VRAIE, seul le bloc d'instructions 2

est exécuté. Si elle est FAUSSE, la condition 3 est évaluée. Si elle est VRAIE, seul le bloc d'instructions 3 est exécuté. Si elle est FAUSSE (c'est à dire qu'aucune des condition précédentes n'est réalisée), seul le bloc d'instructions 4 est exécuté.

Bloc tantque - fintantque

Bloc tantque() - fintantque

Instructions optionnelles de bloc :`arrettantque et suitetantque`
Synonymes respectifs :`while(), endwhile, breakwhile, nextwhile`

Ces instructions permettent l'exécution itérative d'un bloc d'instructions tant que le résultat d'un test est VRAI.
La syntaxe est :

```
tantque(condition)
      liste d'instructions
fintantque
suite du programme
```

L'expression dans l'instruction TANTQUE est évaluée et comparée à l' entier zéro (booléen FAUX), ou n' importe quelle autre valeur (booléen VRAI). Si elle est VRAIE, la liste d'instruction est exécutée jusqu'à l'instruction FINTANTQUE, et le programme se poursuit en boucle avec une nouvelle évaluation de l'instruction TANTQUE.
Dès que l'évaluation de l'expression dans l'instruction TANTQUE devient fausse, le programme se poursuit avec les instructions qui suivent l'instruction FINTANTQUE.

Instructions spécifiques du bloc TANTQUE
Optionnellement, vous pouvez utiliser n'importe où à l'intérieur de ce bloc, l'instruction SUITETANTQUE. Son exécution provoque un débranchement à l'instruction TANTQUE dont elle dépend (S'il y a plusieurs blocs TANTQUE imbriqués, c'est le TANTQUE le plus interne). Utile pour passer facilement un cas que des tests ont traité et rejeté, par exemple.
De même, vous pouvez utiliser à l'intérieur d'un bloc TANTQUE l'instruction ARRETTANQUE. Son exécution provoque une sortie de bloc immédiate, c'est-à-dire un débranchement à l'instruction qui suit le FINTANTQUE du bloc.

```
TANTQUE(condition)
      instructions
      SI (test)
            SUITETANTQUE
      FINSI
      SI (test)
            ARRETTANTQUE
      FINSI
FINTANTQUE
Suite du programme
```


Bloc boucle- finboucle

Bloc boucle()- finboucle

Instructions optionnelles de bloc : `arretboucle et suiteboucle`
Synonymes respectifs : `for(),endfor, breakfor, nextfor`

Ces instructions permettent l'exécution une et une seule fois d' une instruction d' initialisation, l' exécution itérative d'un bloc d'instructions tant que le résultat d'un test est VRAI, et, à la fin de chaque itération mais avant le test de la condition, l' exécution d' une instruction dite de rebouclage. La syntaxe est (version anglophone)

```
for(initiale  ; condition  ; rebouclage)
      liste d'instructions
endfor
```

cette écriture est strictement équivalente à :

```
initiale
while(condition)
      liste d'instructions
     rebouclage
endwhile
```

Exemple (version francophone)

```
boucle(i = 1  ; i <= 10  ; i += 1)
  ...instructions...
finboucle
```

Il y a trois compartiments entre les parenthèses du mot clé `for()` , séparés les uns des autres par un point virgule.

- **1er compartiment** `initiale`

`Initiale` est une (une seule) instruction d' affectation (éventuellement multiple). Cette affectation est exécutée une unique fois à l' entrée dans le bloc.

Par exemple :
`i = 1`
ou la forme multiple :
`i,j = 0,0`

- **2ème compartiment** : `condition`

`Condition` est une expression quelconque dont le résultat est comparé à la valeur entière zéro. Si le résultat est zéro (booléen FAUX) la boucle d' itération s' arrête immédiatement.
Si elle est vraie (tout valeur différente de zéro) , la liste d'instruction est exécutée jusqu' à l' intruction `endfor ou finboucle`. La condition est testée avant toute entrée dans la boucle d' itération, y compris pour la toute première boucle.

- **3ème compartiment** : `rebouclage`

A l' arrivée à l' instruction `endfor ou finboucle`, l' instruction de `rebouclage` est exécutée systématiquement, et ceci avant l' évaluation de la `condition` . Cette instruction est une (une seule) instruction de type affectation (éventuellement multiple).

Par exemple

```
i += 1
```

ou une forme multiple :

```
i,j  = i+1,k+i
```

Puis le programme se poursuit en boucle avec une nouvelle évaluation de la condition prévue dans le compartiment 2

Dès que l'évaluation de la condition devient fausse, la boucle s' arrête et le programme se poursuit avec les instructions qui suivent l'instruction `endfor ou finboucle`.

Instructions spécifiques du bloc `boucle – finboucle`
Optionnellement, vous pouvez utiliser n'importe où à l'intérieur de ce bloc, l'instruction `suiteboucle`. Son exécution provoque un débranchement à l'instruction `boucle` dont elle dépend (S'il y a plusieurs blocs imbriqués, c'est le bloc le plus interne).
De même, vous pouvez utiliser à l'intérieur d'un bloc `boucle` l'instruction `arretboucle`. Son exécution provoque une sortie de bloc immédiate, c'est-à-dire un débranchement à l'instruction qui suit le `finboucle`.

Exemple : (avec des affectations multiples)

```
boucle(i,j = 0,0  ; i <= 10  ; i,j += 1,1)
      instructions
      SI (FonctionX(i))
            suiteboucle
      FINSI
      SI (FonctionY(j))
            arretboucle
      FINSI
finboucle
Suite du programme
```


Bloc liredico - finliredico

Bloc Liredico - FinLiredico

Ces instructions permettent l'exécution itérative d'un bloc d'instructions pour chacun des mots du (ou des) dictionnaires en cours.

Fonction de base

```
Liredico(variable)
      liste d'instructions
FinLiredico
Suite du programme
```

Le dictionnaire est lu depuis son premier mot, qui est chargé dans la `variable` dont le nom est précisé dans l'instruction `LIREDICO`. Puis la liste d'instruction entre les instructions `LIREDICO` et `FinLiredico` est exécutée itérativement jusqu'au dernier mot du dictionnaire, qui est chaque fois chargé dans la `variable` désignée. Après quoi, les instructions suivant `FINLIREDICO` sont exécutées.

Variante `LiredicoEx` (Ex = extended)

```
LiredicoEx(variable; depart; Lgmin  ; Lgmax)
      liste d'instructions
FinLiredico
```

Cette variante de `Liredico` propose trois nouveaux arguments, qui sont trois expressions séparées par des **points-virgules** et dont la valeur est utilisée respectivement pour :

1) Spécifier un mot du dictionnaire à partir duquel la lecture doit commencer. Plus précisément, la lecture du dictionnaire commence au mot exactement spécifié par cette valeur, ou par défaut au mot suivant dans l'ordre alphabétique si la valeur donnée n'est pas un mot valide.
2) Spécifier une longueur minimale. Seuls les mots ayant au moins cette longueur sont sélectionnés. Si l'argument `depart` ne satisfait pas cette contrainte, il n'est pas sélectionné, et le 1er mot effectivement lu sera le 1er mot suivant qui satisfera les contraintes de longueur.
3) Spécifier une longueur maximale. Seuls les mots ayant au plus cette longueur sont sélectionnés. Si l'argument `depart` ne satisfait pas cette contrainte, il n'est pas sélectionné, et le 1er mot effectivement lu sera le 1er mot suivant qui satisfera les contraintes de longueur.

Notez que l'instruction de fin de boucle reste `FinLiredico`, comme dans la variante basique.

Instructions spécifiques du bloc `LIREDICO`
Optionnellement, vous pouvez utiliser n'importe où à l'intérieur de ce bloc, l'instruction `SUITELIREDICO`. Son exécution provoque un débranchement à l'instruction `LIREDICO` dont elle dépend (S'il y a plusieurs blocs `LIREDICO` imbriqués, c'est le `LIREDICO` le plus interne).

De même, vous pouvez utiliser à l'intérieur d'un bloc `LIREDICO` l'instruction `ARRETLIREDICO`. Son exécution provoque un débranchement à l'instruction qui suit le `FINLIREDICO` du bloc, comme si le dernier mot du dictionnaire avait été traité. Si le bloc `LIREDICO` interrompu devait être exécuté une nouvelle fois, il repartirait à nouveau du 1er mot du dictionnaire, et non pas du dernier mot lu lors de la boucle de lecture précédente.

```
LIREDICO(variable)
      instructions
      SI (test)
            SUITELIREDICO
      FINSI
      SI (test)
            ARRETLIREDICO
      FINSI
FINLIREDICO
Suite du programme
```

Note de programmation
Si l'on souhaite utiliser la variable qui contient les mots lus du dictionnaire en dehors du contexte du bloc `LIREDICO-FINLIREDICO` on notera que la valeur de la variable n'est pas changée en sortie de bloc. La dernière valeur affectée persiste donc dans la variable. Cependant, on prendra garde que les lignes de code suivantes conduisent à une erreur d'exécution :

```
LiredicoEx(var;x;y;0)
Finliredico
Message(var)
```

En effet, aucun mot du dictionnaire ne satisfait la contrainte de longueur absurde lgmax = 0. En conséquence aucun mot n'est jamais affecté à la variable var lors de l'exécution du bloc liredico. L'instruction Message provoque donc un arrêt de programme pour cause de `valeur de la variable var inconnue`. Si l'on veut utiliser la variable d'un bloc liredico après l'exécution de ce bloc, il est recommandé d'initialiser la variable à une valeur par défaut avant l'exécution du bloc.
Exemple :

```
var = ""
LiredicoEx(var;x;y;0)
Finliredico
Message(var)
```

Voir Aussi : CHARGERDICO

Bloc lirefichier - finlirefichier

Bloc Lirefichier - Finlirefichier

Instructions optionnelles de bloc :`Arretlirefichier et Suitelirefichier`

Ces instructions permettent l'exécution itérative d'un bloc d'instructions pour chacune des lignes d'un fichier texte.

```
Lirefichier(fichier;enregistrement)
     liste d'instructions
Finlirefichier
```

Le `fichier` dont le nom est précisé en premier argument sera lu ligne à ligne. C'est une expression qui sera interprétée sous forme de chaîne de caractères. Attention elle n'est évaluée qu'à la première exécution de `Lirefichier`, de sorte qu'aucun changement ultérieur dans la valeur de la chaîne de caractères résultat de l'expression `fichier` n'est pris en compte. Il est donc fortement conseillé de n'utiliser qu'une des deux formes simples données dans les exemples qui suivent. En revanche la variable `enregistrement` peut être une variable complexe, c'est à dire par exemple un élément de tableau, car elle est ré-interprétée à chaque exécution.

Le fichier utilisé doit être interprétable sous forme de fichier texte simple, c'est dire une succession de lignes terminées par les caractères de fin de ligne usuels. Utilisez un éditeur de textes de type **Blocnotes** ou **Wordpad**, en prenant la précaution de toujours utiliser un enregistrement sous forme de fichier texte simple (**extension .txt**). Les fichiers créés par Excel, Word, etc.. **ne sont pas** des fichiers textes simples lisibles par EEA.

Lors de l'exécution du programme EEA, chaque ligne est chargée sous forme de chaîne de caractères (sans le caractère de fin de ligne), dans la variable `enregistrement`.

La liste d'instruction entre les instructions `Lirefichier` et `Finlirefichier` est exécutée itérativement jusqu'à la dernière ligne du fichier, dont chaque ligne est chargée dans l'`enregistrement` désigné.

Les routines de bas niveau d'ouverture et de fermeture de fichier sont prises en charge automatiquement

par EEA, de sorte qu'à la fin du script aucun verrou n'est laissé sur le fichier traité.

Exemples
```
i = 1
Lirefichier("c:\\ajl\\mots.txt";ligne)
  Message("ligne %i = %s",i,ligne)
  i += 1
Finlirefichier
```

//notez bien dans cet exemple que la fonction Saisiefichier ne sera exécutée qu'une seule fois.
```
i = 1
Lirefichier(Saisiefichier("Choisissez un nom de
fichier");TableauDeLignes[i])
  Message("ligne %i = %s",i,TableauDeLignes[i])
  i += 1
Finlirefichier
```

Instructions spécifiques du bloc `Lirefichier`
Optionnellement, vous pouvez utiliser n'importe où à l'intérieur de ce bloc, l'instruction `Suitelirefichier`. Son exécution provoque un débranchement à l'instruction `Lirefichier` dont elle dépend (S'il y a plusieurs blocs `Lirefichier` imbriqués, c'est le `Lirefichier` le plus interne).

De même, vous pouvez utiliser à l'intérieur d'un bloc `Lirefichier` l'instruction `Arretlirefichier`. Son exécution provoque un débranchement à l'instruction qui suit le `Finlirefichier` du bloc, comme si le dernier enregistrement du fichier avait été traité. Si le bloc `LIREFICHIER` interrompu devait être exécuté une nouvelle fois, il repartirait à nouveau du début du ficher, et non pas du dernier enregistrement lu lors de la boucle de lecture précédente.

voir aussi
Saisiefichier
caractère d'échappement

Bloc lireliste - finlireliste

Bloc Lireliste - Finlireliste

Instructions optionnelles de bloc : `Arretlireliste et Suitelireliste`
Synonymes respectifs : `forlist(),endforlist, breakforlist, nextforlist`

Ces instructions permettent l'exécution itérative d'un bloc d'instructions pour chacun des éléments d'une liste

```
Lireliste(LISTE;item)
     liste d'instructions
Finlireliste
```

La `LISTE` dont le nom est précisé en premier argument est passée en revue, du premier au dernier élément, et la valeur de chaque élément de liste est affectée à la variable `item`. La variable `item` peut être une variable complexe, c'est à dire par exemple un élément de tableau.

La liste d'instruction entre les instructions `Lirefichier` et `Finlirefichier` est ainsi exécutée itérativement, pour chaque élément de liste.

Exemples

```
ANIMAUX = {"Tigre","Lion","Panthère","Escargot"}
Lireliste(ANIMAUX;animal)
  Message("animal = %s",animal)
Finlireliste
```

Instructions spécifiques du bloc `Lireliste`

Optionnellement, vous pouvez utiliser n'importe où à l'intérieur de ce bloc, l'instruction `Suitelireliste`. Son exécution provoque un débranchement à l'instruction `Lireliste` dont elle dépend (S'il y a plusieurs blocs `Lireliste` imbriqués, c'est le `Lireliste` le plus interne).

De même, vous pouvez utiliser à l'intérieur d'un bloc `Lireliste` l'instruction `Arretlireliste`. Son exécution provoque un débranchement à l'instruction qui suit le `Finlireliste` du bloc, comme si le dernier élément de liste avait été traité. Si le bloc `Lireliste` interrompu devait être exécuté une nouvelle fois, il repartirait à nouveau du début de la liste, et non pas du dernier élément lu lors de la boucle de lecture précédente.

Bloc %xxxx - %retour

Le bloc %xxxx - %retour

Synonyme : `%return`

`%MaFonction()` : Utilisez ce début de bloc pour la définition d' une nouvelle fonction `MaFonction`. La balise de début de bloc commence par le caractère % suivi du nom de la fonction créée, suivie d' une paire de parenthèse entre lesquelles vous définissez la liste, éventuellement vide, des arguments d' entrée. Un argument d' entrée peut prendre n' importe quelle forme légale pour la partie gauche d' une affectation.

Par exemple :

```
%Fonction1()    //fonction sans argument
%Fonction2(a,b) // arguments  : a et b
%Fonction3(a,Tab[1],Tab[a])  //arguments: a Tab[1] Tab[a]
```

`%retour()` : La balise `%retour` est obligatoire, et clôture la définition du code de la fonction. Le mot clé `%retour` doit être suivi d' une paire de parenthèse entre lesquelles vous définissez la liste, éventuellement vide, des valeurs de retour. Ces valeurs peuvent être de simples variables mais aussi des expressions calculées, c'est à dire n' importe quelle forme légale pour la partie droite d' une affectation.

Par exemple

```
%Retour()  //fonction sans argument de retour
%Retour(x) // fonction renvoyant la valeur de x
%Retour(Longueur(mot),x+1) //fonction renvoyant deux valeurs,
résultats de calculs.
```

Instruction optionnelle de bloc
`Exit` : Optionnellement, utilisez le mot clé `Exit` à l'intérieur du bloc `%fonction - %retour` pour déclencher l'exécution immédiate de l'instruction `%retour` et donc la fin de la fonction.

Remarque : `Exit` peut-aussi s'employer dans le programme principal EEA, hors de toute fonction.

Voir aussi
Exit

Sujets liés :

Pourquoi définir une nouvelle fonction
La directive #include
Comment définir une nouvelle fonction
La programmation modulaire, les variables locales
Variables globales
La programmation récursive.
La programmation multi-tâche

Liste des fonctions et opérateurs

Liste des fonctions et opérateurs par typologie

Astuce relative à l'appel à l'aide en ligne dans AJL :
Depuis l'onglet EEA du programme AJL, pour obtenir directement l'aide en ligne relative à une instruction ou une fonction, mettez simplement en surbrillance l'élément sur lequel vous voulez de l'aide, et appuyez sur le **bouton Aide**.

Manipulation de chaînes de caractères

Associe Combine les caractères des éléments d'un tableau en une seule variable
Change Modifie des caractères à des positions données
Comptelettres Compte le nombre d'occurrences de certaines lettres dans une chaîne de caractères
Contient Vérifie si toutes lettres d'une variable sont contenues dans une autre variable
Debut Sélectionne les caractères les plus à gauche d'une variable
Debutinclus Donne la taille du plus long début d'une variable telle que tous les caractères correspondent à une liste de caractères permis
Debutnoninclus Donne la taille du plus long début d'une variable telle qu'aucun des caractères ne corresponde à une liste de caractères interdits
Différence Evalue, de gauche à droite, le nombre de caractères différents entre deux variables
Dissocie Décompose les caractères d'une variable en éléments de tableau d'une seule lettre
Fin Sélectionne les caractères les plus à droite d'une variable
Format Formate les valeurs d'un ensemble de plusieurs variables en une chaîne unique de caractères

Inclus	Vérifie si toues les lettres d'une variable sont constructibles avec un jeu de lettres qui peut contenir des jokers.
Insere	Insère des caractère à partir d'une position donnée
Joker	Récupère, après l'exécution d'une instruction Masque, la valeur des jokers utilisés
Lettre	Transforme un chiffre de 1 à 26 en la lettre correspondant à ce rang dans l'alphabet
Longueur	Calcule la longueur d'une chaîne de caractères
Majus	Transforme en majuscules
Masque	Vérifie si une chaîne de caractères est conforme à un masque comprenant des caractères jokers
Minus	Transforme en minuscules
Optionjoker	Choisit parmi 3 règles de fonctionnement des caractères jokers
Partie	Sélectionne une partie quelconque d'une chaîne de caractères
Poids	Choisit la valeur de chaque lettre, afin de calculer la valeur de mots avec la fonction Valeur
Rang	Transforme une lettre en un nombre de 1 à 26 représentant son rang dans l'alphabet
Retire	Retire des caractères à des positions données
Schulz	Calcule la valeur gématrique d'une chaîne de caractères
Setjoker	Restreint les valeurs que peut prendre un caractère joker dans une fonction Masque
Transpose	Remplace par une autre lettre, dans une variable, un ensemble de lettres choisies
Valeur	Totalise les poids de chaque lettre d'une chaîne de caractères.

Opérateur +	Addition : Concatène deux chaînes de caractères
Opérateur –	(version binaire) Élimine dans une variable toute occurrence d'un ensemble de caractères.
Opérateur –	(version unaire) effectue une inversion droite-gauche des caractères d'une chaîne
Opérateur *	Multiplication : Duplique une chaîne de caractères, le nombre de fois spécifié
Opérateur >>	Décalage : Effectue un décalage à droite, les lettres éjectées sont supprimées.
Opérateur <<	Décalage : Effectue un décalage à gauche, les lettres éjectées sont supprimées.
Opérateur ++	Remplace : Remplace chaque lettre par celle située un certain nombre de rangs plus loin dans l'alphabet
Opérateur <>	Rotation : Effectue un décalage à droite ou à gauche, les lettres éjectées d'un côté réapparaissent de l'autre
Opérateur ::	Index : Sélectionne un caractère par sa position
Opérateur ?	Recherche : recherche à quelles positions se trouvent certains caractères donnés.
Opérateur $+	Union : Complète un argument, autant que nécessaire, des lettres présentes dans un autre argument
Opérateur $-	Soustraction : Retire d'un argument, autant que possible, chaque lettre présente dans un autre argument
Opérateur $&	Intersection : Ne conserve que les lettres présentes dans les deux arguments

---

Fonctions mathématiques

Anp	A(n,p) : nombre d'arrangements sans répétition
Cnp	C(n,p) : nombre de combinaisons sans répétition
Exp	Exponentielle
Fact	Factorielle
Hasard	Un nombre entier au hasard entre 1 et n
Log	Logarithme népérien.

Min	Minimum d'une suite
Max	Maximum d'une suite.
Racine	Racine carrée
Somme	Somme d'une suite

Opérateur +	Addition
Opérateur ++	post incrément
Opérateur ++	pré incrément
Opérateur –	Soustraction.
Opérateur --	post décrément
Opérateur --	pré décrément
Opérateur *	Multiplication.
Opérateur /	Division.
Opérateur %	Modulo
Opérateur ^	Puissance.
Opérateur !	Factorielle d'un nombre.
Opérateur &	ET binaire.
Opérateur |	OU binaire
Opérateur &|	OU exclusif binaire

Fonctions de comparaison et booléennes

Existe — teste l'existence d'une variable ou d'un élément de tableau.

Opérateur < (inférieur)
Opérateur > (supérieur)
Opérateur <= (inférieur ou égal)
Opérateur >= (supérieur ou égal)
Opérateur == (égal)
Opérateur != (différent)
Opérateur && (et logique)
Opérateur || (ou logique)
Opérateur ! (non logique)

Fonctions opérant sur listes

Assemble	rassemble les éléments d'une liste sous forme d'une chaîne de caractères
Change	modifie un élément à une certaine position d'une liste
Dicoliste	crée une liste avec les mots du dictionnaire
Dissocie	range les éléments d'une liste dans un tableau
Insere	insère un ou plusieurs éléments à une certaine position d'une liste
Inter	intersection de deux listes
Lireliste	instruction-bloc de lecture itérative d'une liste
Max	élément maximum d'une ou plusieurs listes
Min	élément minimum d'une ou plusieurs listes
Partie	sélectionne un nombre d'éléments consécutifs d'une liste
Retire	retire certains éléments d'une liste
Separe	sépare une chaîne de caractères en éléments d'une liste
Union	union de deux listes

Opérateur {} (liste)	construit une liste en spécifiant explicitement ses éléments
Opérateur % (série)	construit une liste de nombres consécutifs, commençant par 1
Opérateur % (taille)	donne le nombre d'éléments d'une liste
Opérateur $ (caractères)	construit une liste avec chaque caractère d'une chaîne
Opérateur $ (chaîne)	rassemble les éléments d'une liste dans une seule chaîne de caractères
Opérateur + (ajout)	ajoute les éléments d'une liste à la suite de ceux d'une autre liste
Opérateur - (élimine)	élimine d'une liste tous les éléments présents dans une autre liste
Opérateur - (inverse)	inverse l'ordre des éléments d'une liste
Opérateur * (multiplie)	réplique une liste, ou chaque élément d'une liste
Opérateur :: (index)	sélectionne les éléments d'une liste en fonction de leur index
Opérateur ^ (premier)	sélectionne le premier élément d'une liste
Opérateur ? (recherche)	recherche les éléments d'une liste dans une autre liste
Opérateur & (masque)	sélectionne les éléments d'une liste selon un masque
Opérateur µ (aplatit)	ramène au 1er niveau les éléments de listes imbriqués
Opérateur µ (regroupe)	regroupe par paquets les éléments d'une liste
Opérateur %< (tri)	trie une liste numérique dans l'ordre ascendant
Opérateur %> (tri)	trie une liste numérique dans l'ordre descendant
Opérateur $< (tri)	trie une liste de chaînes dans l'ordre ascendant
Opérateur $> (tri)	trie une liste de chaînes dans l'ordre descendant
Opérateur <> (rotation)	effectue un décalage à droite ou à gauche, les éléments éjectés d'un côté réapparaissent de l'autre
Opérateur << (décale)	effectue un décalage à gauche, les éléments éjectés sont supprimés.
Opérateur >> (décale)	effectue un décalage à droite, les éléments éjectés sont supprimés.

°distribution — distribue un opérateur unaire ou binaire sur les éléments d'une ou plusieurs listes, pour obtenir une liste

réduction° — applique un opérateur binaire sur les éléments d'une seule liste, pour obtenir un seul élément

°produit externe° — distribue un opérateur binaire sur chaque couple possible des éléments de deux listes, pour obtenir une liste

---

Fonctions de changement de type de variable

Chaine
Entier
Flottant
Hexa

Fonctions diverses

Trialpha	Tri des éléments d'un tableau dans l'ordre alphabétique
Trinum	Tri des éléments d'un tableau dans l'ordre numérique
Evalue	Evalue une expression
Exit	Fin de fonction ou fin de programme
Heure	Donne l'heure
Wait	Attend une ou plusieurs valeurs futures

Fonctions en relation avec le dictionnaire AJL

Anagramme	Calcule le nombre d'anagrammes réalisables avec un jeu de lettres

Anagrammeliste Stocke dans une liste les anagrammes réalisables avec un jeu de lettres
Anagrammelistethread Version multi-thread de la fonction précédente
Chargerdico Change le dictionnaire en cours
Debutmot Vérifie si une suite de lettres est un début d'un mot au dictionnaire
Dicoliste Charger tout le dictionnaire dans une liste
Liredico instruction-bloc de lecture itérative d'un dictionnaire
Lexique Vérifie si un mot est constructible avec un lexique donné
Mot Vérifie si une chaîne de caractères est un mot du dictionnaire
Tiragemot Tire au sort un mot du dictionnaire

Fonctions d'affichage, de communication ou de débogage

Breakpoint Point d'arrêt conditionnel
Ecrirefichier Ecrit une ligne dans un fichier
Fermefichier Libère un fichier pour réemploi dans le même script
Lirefichier instruction-bloc de lecture itérative d'un fichier
Message Ecrit une ligne en zone message
Messagebox Ecrit une ligne dans une boîte de message Windows
Pause Suspend momentanément un programme EEA
Saisie Affiche une boîte Windows de saisie d'un texte
Saisiefichier Affiche une boîte Windows de choix d'un fichier
Selection Ecrit une ligne dans la zone liste de mots
Trace Modifie le flag trace
Tracemessage Affiche un texte dans un champ fixe

Chaîne de caractères

Manipulation de chaînes de caractères

Associe Combine les caractères des éléments d'un tableau en une seule variable
Change Modifie des caractères à des positions données
Comptelettres Compte le nombre d'occurrences de certaines lettres dans une chaîne de caractères
Contient Vérifie si toutes lettres d'une variable sont contenues dans une autre variable
Debut Sélectionne les caractères les plus à gauche d'une variable
Debutinclus Donne la taille du plus long début d'une variable telle que tous les caractères correspondent à une liste de caractères permis
Debutnoninclus Donne la taille du plus long début d'une variable telle qu'aucun des caractères ne corresponde à une liste de caractères interdits
Différence Evalue, de gauche à droite, le nombre de caractères différents entre deux variables
Dissocie Décompose les caractères d'une variable en éléments de tableau d'une seule lettre
Fin Sélectionne les caractères les plus à droite d'une variable
Format Formate les valeurs d'un ensemble de plusieurs variables en une chaîne unique de caractères
Inclus Vérifie si toutes les lettres d'une variable sont constructibles avec un jeu de lettres qui peut contenir des jokers.
Insere Insère des caractère à partir d'une position donnée
Joker Récupère, après l'exécution d'une instruction Masque, la valeur des jokers utilisés
Lettre Transforme un chiffre de 1 à 26 en la lettre correspondant à ce rang dans l'alphabet

Longueur	Calcule la longueur d'une chaîne de caractères
Majus	Transforme en majuscules
Masque	Vérifie si une chaîne de caractères est conforme à un masque comprenant des caractères jokers
Minus	Transforme en minuscules
Optionjoker	Choisit parmi 3 règles de fonctionnement des caractères jokers
Partie	Sélectionne une partie quelconque d'une chaîne de caractères
Poids	Choisit la valeur de chaque lettre, afin de calculer la valeur de mots avec la fonction Valeur
Rang	Transforme une lettre en un nombre de 1 à 26 représentant son rang dans l'alphabet
Retire	Retire des caractères à des positions données
Schulz	Calcule la valeur gématrique d'une chaîne de caractères
Setjoker	Restreint les valeurs que peut prendre un caractère joker dans une fonction Masque
Transpose	Remplace par une autre lettre, dans une variable, un ensemble de lettres choisies
Valeur	Totalise les poids de chaque lettre d'une chaîne de caractères.

Opérateur +	Addition : Concatène deux chaînes de caractères
Opérateur –	(version binaire) Élimine dans une variable toute occurrence d'un ensemble de caractères.
Opérateur –	(version unaire) effectue une inversion droite-gauche des caractères d'une chaîne
Opérateur *	Multiplication : Duplique une chaîne de caractères, le nombre de fois spécifié
Opérateur >>	Décalage : Effectue un décalage à droite, les lettres éjectées sont supprimées.
Opérateur <<	Décalage : Effectue un décalage à gauche, les lettres éjectées sont supprimées.
Opérateur ++	Remplace : Remplace chaque lettre par celle située un certain nombre de rangs plus loin dans l'alphabet
Opérateur <>	Rotation : Effectue un décalage à droite ou à gauche, les lettres éjectées d'un côté réapparaissent de l'autre
Opérateur ::	Index : Sélectionne un caractère par sa position
Opérateur ?	Recherche : recherche à quelles positions se trouvent certains caractères donnés.
Opérateur $+	Union : Complète un argument, autant que nécessaire, des lettres présentes dans un autre argument
Opérateur $-	Soustraction : Retire d'un argument, autant que possible, chaque lettre présente dans un autre argument
Opérateur $&	Intersection : Ne conserve que les lettres présentes dans les deux arguments

Associe

Associe(nom,nombre)

La fonction **Associe** collecte un à un les éléments de tableau dont le nom est donné par le premier argument, et les concatène dans l'ordre afin de former la valeur retour. Le premier indice de tableau est 1, et le dernier indice à prendre en compte est spécifié par le second argument.

Syntaxe

Associe(nom,nombre)

Le résultat est de type chaîne de caractère. le premier argument est un nom de tableau, le second un nombre entier

Exemples

```
Tableau[1] = "A"
```

```
Tableau[2] = "ZE"
Tableau[3] = "RTY"
mot = Associe(Tableau,3)
```
le résultat est mot = "AZERTY"

```
a = Saisie("Veuillez taper une suite quelconque de lettres à trier")
x = Dissocie(Tab,a)
Trialpha(Tab,1,x)
a = Associe(Tab,x)
```
Ce petit script demande la saisie d'une suite de lettres, les associe à des éléments de tableau, les trie, puis les ré-associe dans une variable finale.

Voir aussi
Dissocie

Change (chaîne)

Change(chaine,index,valeur)

Cette fonction remplace un et un seul caractère d'une **chaîne**, repéré par son **index**, par un ou plusieurs caractères. Le caractère à remplacer est précisé par l'argument **index**, qui doit être un entier simple représentant le numéro d'ordre, à partir de 1, du caractère concerné. Le ou les caractères de remplacement sont spécifiés dans le 3ème argument **valeur**.Le premier argument **chaine** est toujours une chaîne de caractères, le second **index** un nombre entier, le dernier **valeur** est une chaîne de caractères.

Cette fonction est une extension au domaine des chaînes de caractères de la fonction de même nom, initialement conçue pour le domaine des listes.

Syntaxe
`Change(chaine,index,valeur)`

Exemples
```
ch = Change("abczefg",4,"D")
```
Le résultat est "abcDefg"

```
ch = Change("abcd",2,"BBB")
```
Le résultat est "aBBBcd"

Voir aussi
Change (fonction opérant sur liste)

Comptelettres

Comptelettres(série d'arguments..., chaîne)

Cette jolie fonction compte, dans une chaîne de caractères donnée en dernier argument, le nombre d'occurrence de chacune des lettres spécifiées dans les arguments précédents.

Syntaxe
```
Comptelettres(a,b,c,...,x, chaine)
```
Tous les arguments sont de type chaîne de caractères, de nombre variable.

Les valeurs retours sont de type entier, et en même nombre que les lettres à compter. Chaque valeur retour reçoit le nombre de caractères dans la chaîne test qui correspondent à une des lettres présentes dans le chaîne a (ou b, c, ...)

Exemples
`x = Comptelettres("aeiouy","deux a, quatre e, trois i, deux o, cinq u, un y")`
Le résultat de cette ligne de programme est :
x = 17   (17 occurrences des lettres `"aeiouy"` dans la chaîne `"deux a, quatre e, trois i, deux o, cinq u, un y"`

`nae,nio = Comptelettres("ae","io","deux a, quatre e, trois i, deux o, cinq u, un y")`
Le résultat de cette ligne de programme est :
nae = 6, nio = 7

`na,ne,ni,no,nu,ny = Comptelettres("a","e","i","o","u","y","cinq a, six e, six i, deux o, six u, un y, cinquante quatre lettres au total")`
Les résultats de cette ligne de programme sont :
na = 5
ne = 6
ni = 6
no = 2
nu = 6
ny = 1

étonnant, non ? ;-))

Il est possible de combiner ce type de fonctions à nombre de retours variable avec les fonctions avec nombre d'arguments d'entrée variable. Exemple :

```
maxvoyelles = Max(Comptelettres("a","e","i","o","u","y","cinq a, six e, six i, deux o, six u, un y, cinquante quatre lettres au total"))

Le résultat est simplement
maxvoyelles = 6
```


Contient

Contient(contenant,contenu)

De manière similaire à l'onglet **Contient** de AJL, la fonction **Contient** teste si toutes les lettres d'une chaîne de caractères sont présentes dans un jeu de lettres donné. On précise d'abord la chaîne contenante (la plus grande), puis la chaîne dont ont veut tester qu'elle est contenue (la plus petite). Si une lettre est présente en plusieurs exemplaires dans l'argument contenu, l'argument contenant doit en contenir au moins autant.

La fonction prends en compte le choix effectué dans la case à cocher "Lettres des mots dans le même ordre que dans le jeu de lettres" de l'onglet Contient.

Syntaxe
Contient(contenant,contenu)
Le résultat est de type booléen (1 pour vrai, 0 pour faux), les deux arguments sont de type chaîne.

Exemples
Contient("AZERTYUIOP","ZTA")
le résultat est VRAI.
Contient("AZERTYUIOP","AZTA")
le résultat est FAUX. (il manque un second A)

Voir aussi
Inclus

Debut

Debut(chaine,longueur)

La fonction **Debut** sélectionne la partie gauche d'une chaîne de caractères. Pour cela on précise le nombre de caractères à copier.

Syntaxe
Debut(argument, longueur)

Le résultat est de type chaîne, argument est de type chaîne, longueur un entier.

Exemple
extrait = Debut("AZERTYUIOP",4)
le résultat est "AZER".

Voir aussi
Partie Fin

Debutinclus

Debutinclus(test,reference)

La fonction **Debutinclus** détermine la longueur maximale, depuis le début d'une chaîne de caractères test, telle que tous les caractères rencontrés font partie de la chaîne de caractères de reference. L'ordre des caractères de référence n'a aucune importance.

Ce type de fonction est très utile pour vérifier si une chaîne de caractères correspond bien à un type attendu (par exemple : rien que des consonnes), et si ce n'est pas le cas, déterminer immédiatement la position du premier caractère non conforme.

Syntaxe
Debutinclus(test,reference)

Le résultat est de type entier, les deux arguments de type chaîne.

- Quand le résultat est 0, c'est que le premier caractère de la chaîne test ne se trouve pas parmi ceux de la chaîne reference.
- Si le résultat est supérieur à 0, il indique la longueur maximale, depuis le début de la chaîne test, telle que tous les caractères rencontrés correspondent à l'un des caractères de la chaîne de référence (n'importe lequel).
- Si le résultat correspond à la longueur totale de la chaîne de caractères test, c'est que tous ses caractères font partie de la chaîne test

Exemple
mot = "des Minuscules"
pos = Debutinclus(mot," abcdefghijklmnopqrstuwxyz")

le résultat est 4 (le 5ème caractère de mot, "M", n'est ni un espace ni une minuscule, alors que les 4 premiers le sont).

Note : Pour les programmeurs C, cette fonction correspond à **strcspn**

Voir aussi
Debutnoninclus

Debutnoninclus

Debutnoninclus(test,reference)

La fonction **Debutnoninclus** détermine la longueur maximale, depuis le début d'une chaîne de caractères test, telle que aucun des caractères rencontrés ne fait partie de la chaîne de caractères de reference. L'ordre des caractères de référence n'a aucune importance.

Ce type de fonction est très utile pour vérifier si une chaîne de caractères correspond bien à un type attendu (par exemple : aucune voyelle), et si ce n'est pas le cas, déterminer immédiatement la position du premier caractère non conforme.

Syntaxe
Debutnoninclus(test,reference)

Le résultat est de type entier, les deux arguments de type chaîne.

- Quand le résultat est 0, c'est que le premier caractère de la chaîne test se trouve parmi ceux de la chaîne reference.
- Si le résultat est supérieur à 0, il indique la longueur maximale, depuis le début de la chaîne test, telle qu'aucun des caractères rencontrés ne correspond à l'un des caractères de la chaîne de référence.
- Si le résultat correspond à la longueur totale de la chaîne de caractères test, c'est qu'aucun de ses caractères ne fait partie de la chaîne test.

Exemple
mot = "BCDEFGHKL"
pos = Debutnoninclus(mot,"AEIOUY")

le résultat est 3 (le 4ème caractère de mot, "E", est présent dans "AEIOUY", alors qu'aucun des 3 premiers ne l'est).

Note : Pour les programmeurs C, cette fonction correspond à **strcnspn**

Voir aussi
Debutinclus

Difference

Difference(mot1,mot2)

La fonction **Difference** compte les différences, lettre à lettre et de gauche à droite entre deux chaînes de caractères. Si les deux chaînes ne sont pas de même longueur, la différence de longueur entre en compte dans le calcul de la différence. Si la résultat est zéro, alors les deux chaînes sont identiques.

Syntaxe
Difference(c1,c2)
Le résultat est de type entier, les deux arguments c1 et c2 sont de type chaîne.

Exemples
a = Difference("AZERTY","AWERTY")

le résultat est 1 : une différence avec un W différent d'un Z

b = Difference("AZERTY","AWERTYUIOP")
le résultat est 5 : une lettre différente (W) et 4 lettres en plus (UIOP)

b = Difference("ERTY","AZERTY")
le résultat est 6 : la comparaison est de gauche à droite, aucune des 4 lettres du plus petit argument ne correspond avec l'autre, et le plus grand argument a 2 lettres en plus.

Avertissement de non compatibilité
non compatibilité de résultat avec les versions précédentes, antérieures à AJL 5.0, mais même syntaxe. Anciennement, Difference signalait si les deux mots choisis différaient de moins d'une lettre.

Dissocie (chaîne)

Dissocie(nom,chaîne)

La fonction **Dissocie** collecte un à un les caractères du second argument, considéré comme une chaîne de caractères, et les range dans des éléments de tableau, dont le nom est précisé par le premier argument. Le premier caractère est rangé dans l'indice 1. Le résultat de la fonction est un nombre entier égal à la longueur du second argument, et donc aussi égal au dernier indice de tableau utilisé.

Syntaxe
Dissocie(nom,texte)
Le résultat est de type entier, les deux arguments sont de type chaîne.

Exemples
```
mot = "AZERTY"
x = Dissocie(Tableau,mot)
```
Le résultat est 6 (longueur de "AZERTY"). Par ailleurs, les 6 éléments de tableau suivants ont été affectés :
Tableau[1] = "A"
Tableau[2] = "Z"
Tableau[3] = "E"
Tableau[4] = "R"
Tableau[5] = "T"
Tableau[6] = "Y"

```
a = Saisie("Veuillez taper une suite quelconque de lettres à trier")
x = Dissocie(Tab,a)
Trialpha(Tab,1,x)
a = Associe(Tab,x)
```
Ce petit script demande la saisie d'une suite de lettres, les associe à des éléments de tableau, les trie, puis les ré-associe dans une variable finale.

```
mot = "NORD"
i = 59
x = Dissocie(Tableau[i],mot)
```
Le résultat est 4 (longueur de "NORD"). Par ailleurs, les 4 éléments de tableau suivants ont été affectés :
Tableau[59][1] = "N"
Tableau[59][2] = "O"
Tableau[59][3] = "R"
Tableau[59][4] = "D"

Voir aussi
Associe
Dissocie (liste)

Fin

Fin(chaine,longueur)

La fonction **Fin** sélectionne la partie droite d'une chaîne de caractères. Pour cela on précise le nombre de caractères à copier.

Syntaxe
Fin(argument, longueur)
Le résultat est de type chaîne, argument est de type chaîne, longueur un entier.

Exemple
extrait = Fin("AZERTYUIOP",4)

le résultat est "UIOP".

Voir aussi
Partie Debut

Format

Format(chaîne de formatage,valeur 1, ...,valeur n)

La fonction **Format** permet la création d'une chaîne de caractères incluant la valeur de variables. Chaque variable doit être accompagnée d'une spécification de format qui précise comment elle doit être convertie en chaîne de caractères.

Le principe est le suivant : l'argument "chaîne de formatage" est examiné de gauche à droite, à la recherche d'un spécificateur de format. Tout spécificateur de format commence par le caractère '%'. La première variable de la liste est alors convertie en chaîne de caractères selon le spécificateur trouvé. Le processus se poursuit pour chaque spécificateur trouvé, qui sera associé avec chaque variable de la liste de variables. Il doit y a voir exactement autant de spécificateurs de format que de variables. Par ailleurs, les caractères présents dans la chaîne de formatage et qui ne sont pas des spécificateurs de format sont conservés tels quels.

Les spécificateurs de format sont expliqués par les exemples qui suivent :

spécificateur à tout faire : %z

convertit sous forme de l'équivalent chaîne de caractères. Tous les types de données (entier, décimal, chaîne, listes) peuvent être utilisés.
Si vous ne souhaitez un affichage simple, c'est le seul spécificateur qui vous sera utile. Il n'existe qu'une seule forme, sans variantes :
%z

spécificateur spécialisé chaîne de caractères : %s

%s : crée une chaîne de caractère
variantes:
%10s : crée une chaîne de 10 caractères, blancs à gauche
%-12s : crée une chaîne de 12 caractères, blancs à droite

spécificateur spécialisé nombre entier : %d

%d : nombre entier, représentation décimale ("1")
variantes:
%5d : nombre entier, représentation décimale, sur 5 caractères et les blancs nécessaires à gauche ("   1")

%-5d : nombre entier, représentation décimale, sur 5 caractères et les blancs nécessaires à droite ("1   ")
%05d : nombre entier, représentation décimale, sur 5 caractères et des zéros autant que nécessaires à gauche ("00001")

spécificateur spécialisé nombre avec décimales : %f

%f : nombre avec décimales, affichage par défaut
variantes:
%5.4f : nombre 5 chiffres avant la virgule, 4 après.

spécificateur spécialisé nombre à virgule flottante : %g

%g : nombre flottant, affichage par défaut

ATTENTION : Si vous utilisez un des spécificateurs spécialisés, il est impératif d'utiliser le bon type de spécificateur selon la nature de la variable à afficher. En cas d'erreur de type, la valeur affichée est imprédictible. En cas de doute, utilisez le spécificateur %z à tout faire.

Syntaxe
`Format(chaine) ou Format(chaine,argument1,....,argumentn)`
La fonction **Format** renvoie une chaîne de caractère

Exemple (spécificateur à tout faire)
a = Racine(2)
b = 32
c = "ceci est un test"
d = Format("la valeur de a est %z, b+1 vaut %z, et le texte c est égal à %z",a,b+1,c)
d est alors évalué à : **"la valeur de a est 1.414214, b+1 vaut 33, et le texte c est égal à ceci est un test"**

Exemple (avec des spécificateurs spécialisés):
a = Racine(2)
b = 32
c = "ceci est un test"
d = Format("la valeur de a avec 2 décimales est %1.2f, b+1 vaut %04d, et le texte c est égal à %s",a,b+1,c)
d est alors évalué à : "la valeur de a avec 2 décimales est 1.41, b+1 vaut 0033, et le texte c est égal à ceci est un test"

Remarque
Les fonctions Message Selection et Pause interprètent automatiquement leur(s) argument(s) par appel implicite à la fonction Format.

Note pour les programmeurs C
Format(fmt,x,y,z) est presque identique à l'instruction C sprintf(fmt,x,y,z)

Voir aussi
Chaine
Message

Inclus

Inclus(texte,jeudelettres)

De manière similaire à l'onglet **Inclus** de AJL, la fonction **Inclus** teste si les lettres d'une chaîne de caractères sont constructibles avec un jeu de lettres donné par une autre chaîne de caractères. On précise d'abord la chaîne à vérifier, puis le jeu de lettre.
Ce jeu de lettre peut contenir des caractères jokers "?".

Syntaxe
Inclus(texte, jeudelettres)
Le résultat est de type booléen (1 pour vrai, 0 pour faux), les deux arguments sont de type chaîne.

Exemples
a = Inclus("ZAR","AZERTY")
le résultat est VRAI.

a = Inclus("ZAZY","AZERTY")
le résultat est FAUX (il manque un second Z).

a = Inclus("AZERTY","A?E??Y")
le résultat est VRAI.

Voir aussi
Contient

Insere (chaîne)

Insere(chaine,index,valeur)

Cette fonction insère des caractères dans une chaîne de caractères, à partir d'un index spécifié. Le premier argument **Liste** est chaîne de caractères, le second **index** est un nombre entier simple, le dernier **valeur** est une chaîne de caractères .

Cette fonction est une extension au domaine des chaînes de caractères de la fonction de même nom, initialement conçue pour le domaine des listes.

Syntaxe
`Insere(chaine,index,valeur)`
Le résultat est une **chaîne** où la chaîne spécifiée dans **valeur** vient s'insérer à partir du rang **index** **.**

Exemples:

- Un index égal à 0 provoque une insertion en tout début de liste, l'ensemble des éléments de la liste est décalé à droite.
    - Insere("abc",0,"xy"} --> "xyabc"
- Un index supérieur à 0 provoque une insertion au rang spécifié par l'index. Notamment un index égal au nombre d'éléments insère les nouvelles valeurs juste après le dernier élément.
    - Insere("abc",1,"xy"} --> "axybc"
    - Insere("abc",3,"xy"} --> "abcxy"

Joker

Joker(joker1, joker2, ..., jokerN)

Cette fonction s'utilise exclusivement après l'exécution d'une instruction **Masque** utilisant un ou plusieurs jokers nommés. Elle permet de retrouver les caractères qui dans l'instruction **Masque**, correspondaient aux jokers utilisés.

Voir l'onglet Masque de AJL.

Syntaxe

Joker(J1, J2, ...JN)
La fonction a un nombre d'arguments variables

Le ou les arguments J1, J2, ..., JN de cette fonction correspondent aux jokers recherchés.
"?" ou **"?1"**, ou **"?2"**, ..., **"?26"** pour les jokers d'un seul caractère,
"*" ou **"*1"**, ou **"*2"**, ..., **"*26"** pour les jokers de plusieurs (éventuellement aucun) caractères,
"+" ou **"+1"**, ou **"+2"**, ..., **"+26"** pour les jokers de plusieurs (au moins 1) caractères

Les résultats rendus sont les caractères qui ont été "masqués" par les jokers. Si l'instruction **Masque** précédente a échoué, ou si le joker demandé est invalide, le résultat est une chaîne vide. Il y autant de valeurs de retour que d'arguments fournis

Exemple :
```
Setjoker(1)
Setjoker(2)
Setjoker(3)
Setjoker(4)
m = Masque("mlkdjf(216)kjldfq","+1(*2)*3?4")
a,b,c,d = Joker("*2","?4","+1","*3")
```

Les 4 instructions `Setjoker` autorisent les jokers nommés 1 2 3 4 à masquer n'importe quel caractère.

l'instruction `Masque` utilise 4 jokers dans le masque complexe `"+1(*2)*3?4"` pour trouver :
- n'importequelle chaîne d'au moins un caractère (effet de **"+1"**)
- une parenthèse ouvrante '('
- n'importequelle chaîne (effet de **"*2"**)
- une parenthèse fermante ')'
- n'importequelle chaîne (effet de **"*3"**)
- un unique caractère final (effet de **"?4"**)

le tout dans la chaîne de caractère cible `"mlkdjf(216)kjldfq"`

- le joker **+1** correspond donc à "`mlkdjf`"
- le joker ***2** correspond donc à "`216`"
- le joker ***3** correspond donc à "`kjldf`"
- le joker **?4** correspond donc à "`q`"

Finalement, les variables a, b, c, d prennent respectivement les valeurs `"216", "q" , "mlkdjf" , "kjldf".`

Voir aussi
fonction Masque
onglet Masque
Optionjoker
Setjoker

Lettre

Lettre(nombre)

La fonction **Lettre** donne la lettre correspondant à un rang donné dans l'ordre alphabétique. Si le nombre n'est pas entre 1 et 26, le résultat est "" (chaîne vide). Sinon, c'est une lettre de A à Z, toujours en majuscules, pour les nombres de 1 à 26.

Syntaxe
Lettre(rang)

Le résultat est de type chaîne, l'argument de type entier

Exemples
Lettre(3) est égal à "C".
Lettre("C") est égal à "".
Lettre(0) est égal à "".

Voir aussi
Rang

Longueur

Longueur(chaine)

La fonction **Longueur** détermine la longueur d'une chaîne de caractères passée en argument.

Syntaxe
Longueur(argument)

Le résultat est de type entier, l'argument est de type chaîne.

Exemples
```
mot = "AZERTY"
lg1 = Longueur(mot)
mot = ""
lg2 = Longueur(mot)
lg3 = Longueur("Cette phrase compte trente lettres")
```

Les résultats respectifs sont : 6, 0, 34. (ben oui, il y a 4 espaces. ;-))

Majus

Majus(chaine)

La fonction **Majus** convertit en majuscules les minuscules présentes dans une chaîne de caractères.

Syntaxe
Majus(argument)

Le résultat est de type chaîne, argument est de type chaîne.

Exemple
maj = Majus("aàâäbcçdeéèêëfghiîïjklmnoôöpqrstuùûüwxyz")
le résultat est "AAAABCCDEEEEEFGHIIIJKLMNOOOPQRSTUUUUVWXYZ".

Voir Aussi
Minus

Masque

Masque(chaine,masque)

La fonction **Masque**, tout comme l'onglet masque, sélectionne les mots qui correspondent à un masque constitué de lettres et des caractères jokers "?" (1 lettre), "+" (1 ou plusieurs lettres) et "*" (zéro ou plusieurs lettres)

Tout comme l'onglet masque ou le filtre masque de l'onglet Expert, vous pouvez nommer vos jokers en les faisant précéder par un chiffre de 1 à 26.

Les particularités des jokers nommés ?1 à ?26 de l'onglet masque sont ici aussi utilisables, et le choix de paramètre intitulé dans cet onglet "Règles concernant les caractères jokers ?1 à ?26" est respecté.
Attention, si vous utilisez plusieurs Masques à la suite, et que ces filtres partagent un de ces jokers, les règles concernant les caractères jokers répétés s'appliquent aussi ! Le choix de paramètre qui dicte le fonctionnement de ces jokers peut se régler soit dans l'onglet masque commé déjà indiqué, mais aussi par l'instruction Optionjoker

Syntaxe
Masque(mot,masque)
Le résultat est de type booléen, les deux arguments sont de type chaîne.

Exemples
```
Masque("AZERTY","A?E?TY")
```
le résultat est VRAI, car les jokers ? et ? correspondent à "Z" et "R" respectivement

```
Masque("AZERTY","A*?Y")
```
le résultat est VRAI, car le joker * correspond à "ZER", et le joker ? à "T"

```
Masque("AZERTY","B+")
```
le résultat est FAUX, car AZERTY ne commence pas par B, de sorte que le joker "+" n'a pas de correspondance possible.

```
Masque("AZERTY","A*10RT?")
```
le résultat est VRAI, et le joker nommé *10 correspond à "ZE". Puisque ce joker est nommé, et repéré par l'indice 10, sa valeur "ZE" pourrait être récupérée par une instruction Joker("*10") ultérieure.

Utilisation de Masque avec des caractères absents du dictionnaire
Il est essentiel de comprendre que les jokers d'un masque sont limités à un jeu de caractères. Ce jeu de caractère est soit spécifié de manière explicite avec la fonction Setjoker, soit de manière implicite. Dans ce dernier cas, les valeurs possibles sont l'ensemble des caractères présents dans les mots du dictionnaire en cours.
Ainsi, si vous souhaitez dans votre script masquer des chiffres ou des minuscules par exemple, il est nécessaire de définir au préalable les valeurs éligibles par Setjoker.
Exemple :
```
Setjoker("ABCDEF12345",0)
a = Masque("A12B","A*B")
```

le résultat est vrai, car * a été autorisé explicitement à masquer, entre autres, les chiffres 1 et 2. Sinon, le résultat eût été FAUX.

Voyez aussi
Joker
Optionjoker
Setjoker

Minus

Minus(chaine)

La fonction **Minus** convertit en minuscules les majuscules présentes dans une chaîne de caractères.

Syntaxe
Minus(argument)

Le résultat est de type chaîne, argument est de type chaîne.

Exemple
min = Minus("AZERTY")

le résultat est "azerty".

Voir Aussi
Majus

Optionjoker

Optionjoker(nombre)

Cette fonction pilote le fonctionnement des jokers nommés de type ?1, ?2, ..., ?26, dans le cas où vous utiliseriez plusieurs fois le même joker nommé dans un même masque. C'est exactement le même réglage que celui proposé dans l'onglet Masque, où vous trouvez trois boutons correspondant aux trois choix possibles.

L'argument de cette instruction doit être un nombre entier valant 0,ou 1, ou 2. Les trois valeurs ont les effets suivants :

- Optionjoker(0) : Pas de règle particulière.

Il n'y a aucune contrainte d'utilisation de ces jokers. Dans ce mode, un joker nommé "?1" par exemple a exactement le même fonctionnement qu'un joker simple "?", tel que décrit dans le fonctionnement simple du filtre Masque ou de la fonction Masque

- Optionjoker(1) : Un joker répété doit toujours masquer la même lettre.

- Optionjoker(2) : A un joker correspond une seule lettre, et réciproquement

Comme la règle précédente, mais en plus une lettre ne peut pas se retrouver dans deux jokers différents.

Syntaxe
Optionjoker(valeur)
La fonction ne renvoie aucune valeur en retour.

Exemples
Optionjoker(1)

Un masque "?1LL?1" peut trouver seulement ALLA et ELLE; alors que ?LL? (ou le mode "Pas de règle particulière) aurait aussi trouvé ALLO.
Un masque "?1?2*?1?2" peut servire à repérer les mots qui commencent et finissent par les mêmes deux lettres : tels : ALLUVIAL, ONCTION, SEMEUSE, etc...

Optionjoker(2)

Un masque "E?1?2E" trouve ELFE, ETRE etc..., mais rejette ELLE car le 'L' ne peut être à la fois dans ?1 et ?2.

Voir aussi
fonction Masque
onglet Masque
Joker
Setjoker

Partie (chaîne)

Partie(chaine,position,longueur)

La fonction **Partie** sélectionne une partie d'une chaîne de caractères. Pour cela on précise la chaîne de caractère, la position du premier caractère à sélectionner (valeur 1 pour le premier caractère), et une longueur à sélectionner.

Syntaxe
Partie(argument, position, longueur)

Le résultat est de type chaîne, argument est de type chaîne, position est un entier, longueur un entier.

Exemple
extrait = Partie("AZERTYUIOP",2,3)
le résultat est "ZER".

Voir aussi
Debut
Fin
Opérateur ::

Poids

Poids(...suite de 26 valeurs...)

La fonction **Poids** permet d'affecter un poids à chaque lettre de l'alphabet, ce qui permet ensuite de calculer une "valeur" d'un mot, comme somme des poids de ses lettres, ce qui est l'objet de la fonction Valeur.
Par défaut les poids de chaque lettre sont celles déjà choisies dans le menu Options -> Valeur des lettres. La fonction Poids permet donc de modifier ces valeurs de référence, mais toutefois, ces modifications resteront locales au programme EEA. A la fin du programme, les valeurs préalablement choisies dans le menu Options -> Valeurs sont restituées.

Syntaxe
Poids (a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z)
Où a,b,...,z sont des expressions, dont le résultat doit être un nombre entier qui sera la valeur des 26 lettres de l'alphabet. Il n'y a pas de valeur retour.

Exemple
```
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26)
```
affecte à chaque lettre une valeur égale à son rang dans l'alphabet. Dans ces conditions :
```
x = Valeur("DEUX CENT CINQUANTE HUIT")
```
la valeur de x est alors **258**. Tiens tiens tiens...

Voir aussi

[Valeur]

Rang

Rang(lettre)

La fonction **Rang** détermine le rang dans l'ordre alphabétique du **premier** caractère d'une chaîne de caractères passée en argument. Si ce n'est pas une lettre, le résultat est 0. Sinon, c'est un nombre de 1 à 26 pour les lettres de A à Z. Le caractère dont le Rang est calculé peut être indifféremment en minuscules ou en majuscules.

Syntaxe
Rang(lettre)

Le résultat est de type entier, l'argument de type chaîne.

Exemples
Rang("C") est égal à 3.
Rang("a") est égal = 1.
Rang("3") est égal à 0.
Rang(" ") est égal à 0.

Voir aussi
Lettre

Retire (chaîne)

Retire(chaine,index)

Cette fonction supprime un ou plusieurs caractères d'une chaîne. Les caractères à supprimer sont précisés par l'argument index, qui peut être soit un entier simple, soit une liste d'entiers, correspondant aux rangs (à partir de 1) des caractères concernés.

Cette fonction est une extension au domaine des chaînes de caractères de la fonction de même nom, initialement conçue pour le domaine des listes.

Les index inférieurs strictement à 1 ou supérieurs strictement au nombre d'éléments de la chaîne de caractères sont simplement ignorés.

Syntaxe
`Retire(chaine,index)`
Le résultat est la chaîne de caractères **`chaine`** dont on a retiré les caractères spécifiés par **`index`**.

Exemples
```
A = "azerty"

B = Retire(A,4)
le résultat est "azety"

B = Retire(A,{9,1,5,3,1} )
le résultat est "zry" (caractères 1,3,5 = "a", "e", "t" retirés, index 9
ignoré, second index 1 ignoré
```

Schulz

Schulz(chaine)

La fonction **Schulz** détermine la valeur gématrique d'une chaîne de caractères, obtenue par addition de la valeur de chacune des lettres ou chiffres, selon les règles suivantes :
- valeur d'une lettre, minuscule ou majuscule = son rang dans l'alphabet
- valeur d'un chiffre = sa valeur numérique
- valeur de tout autre caractère = 0

Syntaxe
Schulz(argument)
Le résultat est de type entier, l'argument est de type chaîne.

Exemples
```
mot = "AZERTY"
x = Schulz(mot)
```
la valeur est 95 ("A" = 1, "Z" = 26, etc..)

```
mot = "deux CENT VINGT deux"
x = Schulz(mot)
```
la valeur de x est alors **222**. Amusant n'est ce pas ?...

```
mot = "abc & 123"
x = Schulz(mot)
```
la valeur de x est alors **12** ("a" = 1, "b" = 2, "c" = 3, "&" = 0, "1" = 1, "2" = 2, "3" = 3)

Voir Aussi
Valeur

http://remi.schulz.perso.neuf.fr/

Setjoker

Setjoker(valeurs,nombre)

Cette fonction s'utilise avant l'exécution d'une instruction **Masque** utilisant un ou plusieurs jokers nommés de type ?1, ?2, ...etc..., ?26, ou +1, +2, ...etc..., +26, ou *1, *2, ...etc..., *26. Elle permet de restreindre à un ensemble de caractères les trois types de jokers "?" "+" ou "*" qui portent le numéro spécifié dans l'argument nombre.
Ce mécanisme est le même que celui mis en œuvre dans le "filtre" JOKER de l'onglet de recherche Expert.

Il est parfois utile de forcer un joker nommé à accepter n'importe quel caractère. Il suffit alors d'omettre l'argument valeurs (cf syntaxe2 ci-dessous)

Syntaxe1
```
Setjoker(valeurs,nombre)
```

Le 1er argument est une chaîne de caractère représentant les valeurs aux quelles devront se restreindre les jokers dont le numéro est spécifié dans le second argument de type numérique. Il n'y a pas de valeur retour. Les jokers non nommés (? ou *, sans plus de précision) correspondent à la valeur 0 de l'argument nombre.

Syntaxe2
`Setjoker(nombre)`

Dès lors, le joker de numéro `nombre` pourra prendre n'importe quelle valeur.

Exemples :

```
Setjoker("ABCDEF",1)
Setjoker("GHIJKL",2)
test = Masque(mot,"?1?1*2)
```

Dans cet exemple, pour que la valeur rendue par la fonction Masque soit à VRAI, il faut :
1) que les deux premiers caractères du mot soient deux lettres présentes dans "ABCDEF" ; effet des deux jokers ?1 consécutifs
2) que l'ensemble des caractères suivants, quel que soit leur nombre, soient des lettres présentes dans "GHIJKL"; effet du joker *2

```
Setjoker(11)
Setjoker("0123456789",12)
Setjoker(13)

test = Masque("mE45Zlùml(142857)mldvcùm78jHJH","*11(*12)*13")
Selection(Joker("*12"))
```

Dans cet exemple, on recherche `"*11(*12)*13"` c'est à dire compte tenu des 3 instructions `Setjoker` précédentes, n'importe quoi (le joker 11) suivi, entre deux parenthèses, d'un champ numérique (le joker 12) suivi de n'importe quoi (le joker13). L'instruction `Selection` affiche donc la valeur 142857.

Voir aussi
Joker
fonction Masque
onglet Masque
Optionjoker

Transpose

Transpose(cible,anciens,nouveaux)

La fonction **Transpose** permet de substituer dans une chaîne de caractères des lettres par d'autres. Pour celà on précise 3 arguments : le premier est la chaîne à modifier, le second précise la suite de lettres pour lesquelles une substitution est désirée, le troisième la suite des lettres à substituer, en correspondance lettre à lettre avec l'argument 2. Les deux derniers arguments doivent avoir impérativement la même longueur.

Syntaxe
Transpose(mot,ancien,nouveau)
Tous les arguments sont de type chaine.

Exemple
Transpose("BABA COOL","ABCDE","FGHIJ")

demande à remplacer dans "BABA COOL" :

- les A par des F,
- les B par des G,

- les C par des H,
- les D par des I,
- les E par des J
- les autres lettres sont inchangées.

De sorte que le résultat final est "GFGF HOOL"

Valeur

Valeur(chaine)

La fonction **Valeur** détermine la valeur d'une chaîne de caractères, obtenue par addition de la valeur de chacune des lettres, telle que paramétrée dans la fenêtre Paramètres\Valeur des lettres. ou par l'instruction Poids. **Attention**, seules les lettres en majuscules sont prises en compte.

Syntaxe
Valeur(argument)
Le résultat est de type entier, l'argument est de type chaîne.

Exemples
```
mot = "AZERTY"
x = Valeur(mot)
```
selon les règles du scrabble standard, la valeur est 24

```
mot = "AzERTY"
x = Valeur(mot)
```
la valeur est 14, car le z minuscule compte pour zéro

```
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26)
x = Valeur("DEUX CENT VINGT DEUX")
```
la 1ère instruction affecte à chaque lettre une valeur égale à son rang dans l'alphabet.
la valeur de x est alors **222**. Tiens tiens tiens...

Voir Aussi
Poids
Schulz

Opérateur + (concaténer)

Opérateur +

L'opérateur **+** concatène deux chaînes de caractères. Cet opérateur agit en concaténation dès qu'un des deux arguments à droite ou à gauche est de type chaîne. Si ce n'est pas le cas, il est interprété comme l'opérateur d'addition entre nombres entiers ou décimaux.

Syntaxe
chaine1 **+** chaine2

Le résultat est de type chaîne, l'un au moins des deux arguments doit être de type chaîne.

Exemples

somme = "AZERTY" **+** "UIOP"
le résultat est "AZERTYUIOP"

nombre = 11 * 3
somme = "AZ" **+** nombre
le résultat est "AZ33"

Voir aussi
Opérateur + (addition)
opérateur + (addition de listes)

Opérateur - (éliminer tous)

Opérateur -

L'opérateur **-** élimine entièrement d'une chaîne de caractère (opérande gauche) tout occurrence d'une lettre présente dans l'opérande droit.

Syntaxe
chaine1 **-** chaine2

Le résultat est de type chaîne, l'un au moins des deux arguments doit être de type chaîne. Si ce n'est pas le cas, l'opérateur est interprété comme la soustraction entre deux nombres.

Exemple
x = "ELIMINATION" **-** "NI"
le résultat est "ELMATO", qui correspond à l'argument 1 "ELIMINATION" duquel on a ôté toutes les lettres "N" et "I".

Voir aussi
Opérateur + (addition)

opérateur - (élimination d'élements de liste)

version à un seul argument (unaire)
Cet opérateur **-** possède une autre signification lorsqu'il est utilisé sur un seul argument comme dans
x = - "AZERTY"
opérateur - (inverser)

Opérateur - (inverser)

Opérateur -

L'opérateur **-** inverse l'ordre des lettres d'une chaîne de caractères. C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
- chaine

L'argument et le résultat sont de type chaîne.

Exemple

x = **-** "TITRES"
le résultat est "SERTIT".

voir aussi
opérateur - (inversion de liste)
Cet opérateur **-** possède une autre signification lorsqu'il est utilisé avec deux arguments comme dans
x = "AZERTY" - "ZER"
opérateur - (éliminer)

Opérateur * (multiplier)

Opérateur *

L'opérateur ***** multiplie une chaîne de caractères par un nombre. Le nombre et la chaîne peuvent être indifféremment l'argument droite ou l'argument gauche. Le résultat est le répétition de la chaîne de caractères, le nombre de fois précisé par l'argument nombre.

Cet opérateur agit ainsi dès qu'un seul des deux arguments à droite ou à gauche est de type chaîne. Si ce n'est pas le cas, il est soit interprété comme l'opérateur de multiplication entre nombres entiers ou décimaux, soit rejeté en erreur dans le cas d'une tentative de multiplication de deux chaînes entre elles.

Syntaxe
`nombre * chaine` ou bien `chaine * nombre`
Le résultat est de type chaîne, l'un des deux arguments doit être de type chaîne, l'autre de type entier.

Exemple
x = "AJL" *** 3**
le résultat est "AJLAJLAJL"

Voir aussi
opérateur * (multiplication de listes)

Opérateur ^ (premier)

Opérateur ^

L'opérateur **^** sélectionne le premier élément d'une chaîne. Il est unaire à droite. Il est strictement équivalent à l'appel à l'opérateur index, avec l'argument 1 à droite.

Syntaxe
^a
Le résultat est le premier caractère de la chaîne a

Exemples
```
a = "machin"
i = ^a
```

i est alors égal à la chaîne d'un seul caractère "m"

Opérateur :: (index)

Opérateur ::

L'opérateur index **::** sélectionne un caractère d'une chaîne de caractères. Pour cela on précise en premier argument (gauche) la chaîne de caractère, et en second argument (droite) la position du caractère à sélectionner. Une position 1 sélectionne le premier caractère.

L'argument de droite peut aussi être une liste. Dans ce cas le résultat est une chaîne composée des caractères sélectionnés.

Syntaxe
chaine**::**position
ou
chaine**::**INDEX

Le résultat est de type chaîne, l'argument chaine est de type chaîne, l'argument position est un entier.

Exemples
extrait = "AZERTYUIOP" **::** 2
le résultat est "Z", car Z est le second caractère de AZERTYUIOP.

extrait = "AZ" **::**99
le résultat est "" (chaîne vide)

autre = "AZERTYUIOP" **::** {2,1,2,1}
le résultat est "ZAZA"

Voir aussi
Debut
Fin
Partie
Opérateur :: (index) sur listes

Opérateur ? (recherche)

Opérateur ?

L'opérateur ? est un opérateur de recherche dans une chaîne de caractères. Le résultat est une liste.

Syntaxe
`chaine?ref`

Les deux opérandes sont des chaînes de caractères. L'opérateur recherche les occurrences de la chaîne de droite **`ref`** dans la liste de gauche **`chaine`**. Le résultat est une liste d'index, tels qu'à ces positions dans la chaîne de caractères, on y trouve exactement la chaîne **`ref`** recherchée. Si aucun élément ne correspond, la liste rendue est vide.

Exemples

```
ch = "rechercher les lettres e"

ch::"e" --> {2,5,9,13,17,21,24}
ch::"le" --> {12,16}
```

Cet opérateur devient une meilleure alternative à la fonction Position, qui est désormais mentionnée "obsolète" dans la documentation mais maintenue pour compatibilité... jusqu'à ce que j'en décide autrement !

Voir aussi
opérateur ? (recherche sur listes)

Opérateur ++ (transpose)

Opérateur ++

Transposition des caractères d'un mot (argument de gauche, de type chaîne) par remplacement de chacune de ses lettres par celle placée un certain le nombre de fois plus loin dans l'ordre alphabétique, le nombre en question est précisé par l'argument de droite, de type entier. Un argument négatif provoque un décalage dans l'autre sens.

Les 26 lettres "cyclent" sur elles-mêmes, de sorte qu'un "Z" transposé une fois devient un "A". Les caractères en minuscules tout comme ceux en majuscules sont transposés. Les caractères qui ne seraient pas des lettres demeurent inchangés.

Syntaxe
chaine **++** nombre

Le résultat est de type chaîne, l'argument gauche aussi, et l'argument droite de type entier.

Exemples
x = "ABCD" **++** 1
le résultat est "BCDE"

x = "AZERTY" **++** 2
le résultat est "CBGTVA"

x = "AZERTY" **++** -1
le résultat est "ZYDQSX"

Opérateur >> (décale droite)

Opérateur >>

Cet opérateur décale d'un ou plusieurs crans (argument de droite) à droite les caractères d'un mot (argument de gauche, de type chaîne). Les caractères décalés sont purement et simplement éliminés, à la différence de l'opérateur rotation qui les fait cycler. De sorte que la longueur de la chaîne de caractère résultat est toujours diminuée du nombre de décalages effectués.

Un décalage d'un nombre supérieur ou égal à la longueur de la chaîne initiale produit toujours une chaîne vide. Un décalage négatif est sans effet.

Syntaxe
chaine **>>** nombre

Le résultat est de type chaîne, l'argument gauche aussi, et l'argument droite de type entier.

Exemples

x = "AZERTY" **>>** 2

le résultat est "AZER"

Avertissement de non compatibilité
Dans les versions inférieures à 5.0, cette fonction correspondait à l'opérateur ++ actuel.

Opérateur << (décale gauche)

Opérateur <<

Cet opérateur décale d'un ou plusieurs crans (argument de droite) vers la gauche les caractères d'un mot (argument de gauche, de type chaîne). Les caractères décalés sont purement et simplement éliminés, à la différence de l'opérateur rotation qui les fait cycler. De sorte que la longueur de la chaîne de caractère résultat est toujours diminuée du nombre de décalages effectués.

Un décalage d'un nombre supérieur ou égal à la longueur de la chaîne initiale produit toujours une chaîne vide. Un décalage négatif est sans effet.

Syntaxe
chaine **<<** nombre

Le résultat est de type chaîne, l'argument gauche aussi, et l'argument droite de type entier.

Exemples

x = "AZERTY" **<<** 2
le résultat est "ERTY"

Opérateur <> (rotation)

Opérateur <>

Fait tourner les caractères d'un mot (opérande de gauche) par décalage à droite, le nombre de fois indiqué par l'opérande de droite. Les caractères qui "sortent" à droite rentrent par la gauche. On tourne dans l'autre sens avec des valeurs négatives.

Syntaxe
chaine **<>** nombre

Le résultat est de type chaîne, l'argument gauche aussi, et l'argument droite de type entier.

Exemples
"ABCD" **<>** 1 = "DABC"
"AZERTY" **<>** 2 = "TYAZER"

Et avec des valeurs négatives :
"ABCD" **<>** -1 = "BCDA"
"AZERTY" **<>** -2 = "ERTYAZ"

Opérateur $+ (union)

Opérateur $+

il s'agit d'un des 3 opérateurs ensemblistes, fonctionnant sur chaînes de caractères.

Les deux opérandes d'une opération a **$+** b sont considérés comme des ensembles de lettres. Le résultat est l'opération est l'ensemble de gauche complété du plus petit sous-ensemble de lettres extraites de l'opérande de droite, tel que l'opérande de droite puisse être inclus dans l'ensemble résultat.

Dit autrement, cet opérateur se distingue d'une simple concaténation (opérateur +) par le fait qu'une lettre qui serait présente dans les deux arguments, n'est présente qu'une seule fois dans le résultat final.

L'échange droite - gauche des deux opérandes change l'ordre mais pas le contenu du résultat.

Syntaxe
chaine **$+** chaine

Le résultat est de type chaîne, les deux arguments aussi

Exemple

a = "abbcd"
b = "aabce"
r = a **$+** b

le résultat r est "bdaabce"

En effet, il suffit de compléter a = "abbccd" d'un "a" supplémentaire (car b en contient 2) et d'un "e" (car a n'en contient aucun) pour que toutes les lettres de b = "aabce" soient incluses dans le résultat

voir aussi
Opérateur $-
Opérateur $&

A propos des 3 opérateurs ensemblistes
a == (a $& b) + (a $- b)
a $+ b == (a + b) $- (a $& b)

Opérateur $- (retirer)

Opérateur $-

il s'agit d'un des 3 opérateurs ensemblistes, fonctionnant sur chaînes de caractères.

Les deux opérandes d'une opération a **$-** b sont considérés comme des ensembles de lettres. Le résultat est l'opération est l'argument de droite, duquel on a ôté chacune des lettres de l'argument de gauche.

Syntaxe
chaine **$-** chaine

Le résultat est de type chaîne, les deux arguments aussi

Exemple

a = "abbcd"
b = "aabce"
r = a **$-** b
le résultat r est "bd"

En effet, la seule lettre "a" de l'ensemble a est retirée par une des deux lettres "a" de l'ensemble b; une des 2 lettres "b" par l'unique lettre "b" de l'ensemble b, reste donc une lettre "b" dans le résultat; la lettre c est éliminée, la lettre "d" reste car l'ensemble b n'en compte pas.

Notez bien la différence avec l'opérateur - (retire)
a = "abbcd"
b = "aabce"
r = a **-** b
le résultat r est "d", car tous les "a", tous les "b", tous les "c" et tous les "e" sont retirés de a.

voir aussi
Opérateur $+
Opérateur $&
Opérateur -

A propos des 3 opérateurs ensemblistes
a == (a $& b) + (a $- b)
a $+ b == (a + b) $- (a $& b)

Opérateur $& (intersection)

Opérateur $&

il s'agit d'un des 3 opérateurs ensemblistes, fonctionnant sur chaînes de caractères.

Les deux opérandes d'une opération a **$&** b sont considérés comme des ensembles de lettres. Le résultat est l'ensemble des lettres communes aux deux arguments(intersection de a et b, en termes mathématiques).

Cet opérateur est très utile pour vérifier, par exemple, que deux mots n'ont aucune lettre en commun. Il suffit en effet de vérifier que leur intersection est vide.

Syntaxe
chaine **$&** chaine

Le résultat est de type chaîne, les deux arguments aussi

Exemple

a = "abbcd"
b = "aabce"
r = a **$&** b

le résultat r est "abc"

Seules ces 3 lettres se trouvent à la fois dans l'ensemble a et dans l'ensemble b

voir aussi
Opérateur $+

Opérateur $-

A propos des 3 opérateurs ensemblistes
a == (a $& b) + (a $- b)
a $+ b == (a + b) $- (a $& b)

Mathématiques et logiques

Fonctions mathématiques et logiques

Anp	A(n,p) : nombre d'arrangements sans répétition
Cnp	C(n,p) : nombre de combinaisons sans répétitions
Exp	Exponentielle
Fact	Factorielle
Hasard	Un nombre entier au hasard entre 1 et n
Log	Logarithme népérien.
Min	Minimum d'une liste
Max	Maximum d'une liste.
Racine	Racine carrée
Somme	Somme d'une liste

Opérateur +	Addition
Opérateur ++ (post incrément)	
Opérateur ++ (pré incrément)	
Opérateur -- (post décrément)	
Opérateur -- (pré décrément)	
Opérateur –	Soustraction.
Opérateur *	Multiplication.
Opérateur /	Division.
Opérateur %	Modulo
Opérateur ^	Puissance.
Opérateur &	ET binaire.
Opérateur |	OU binaire
Opérateur &|	OU exclusif binaire
Opérateur ! (factorielle)	

Existe	teste l'existence d'une variable ou d'un élément de tableau.

Opérateur < (inférieur)
Opérateur > (supérieur)
Opérateur <= (inférieur ou égal)
Opérateur >= (supérieur ou égal)
Opérateur == (égal)
Opérateur != (différent)
Opérateur && (et logique)
Opérateur || (ou logique)
Opérateur ! (non logique)

Anp (arrangements)

Anp(n,p)

La fonction **Anp(n,p)** calcule le nombre d'**arrangements** sans répétition de p objets choisis dans un ensemble de n objets. Elle est souvent notée A_n^p en mathématiques. En utilisant la notation factorielle, on a l'identité suivante :

Anp(n,p) = n! / (n - p)!

Syntaxe
a = Anp(nombre,nombre)

Le résultat est un nombre entier, les deux arguments aussi.

Exemple

a = Anp(4,2)

résultat : a = 12. Il y a en effet 12 **arrangements** de 2 choix parmi 4 objets a b c d
ab ba ac ca ad da bc cb bd db cd dc

Voir aussi
Cnp
Fact

Cnp (combinaisons)

Cnp(n,p)

La fonction **Cnp(n,p)** calcule le nombre de **combinaisons** sans répétition de p objets choisis dans un ensemble de n objets. Elle est souvent notée C_n^p en mathématiques. En utilisant la notation factorielle, on a l'identité suivante :

Cnp(n,p) = n! /(p! (n - p)!)

Syntaxe
c = Cnp(nombre,nombre)

Le résultat est un nombre entier, les deux arguments aussi.

Exemple

c = Cnp(4,2)

résultat : c = 6. Il y a en effet 6 **combinaisons** de 2 choix parmi 4 objets a b c d
ab ac ad bc bd cd

Voir aussi
Anp
Fact

Fact (factorielle)

Fact(n)

La fonction **Fact(n)** est la factorielle de n. Elle calcule le nombre de **permutations** sans répétition de n objets entre eux. La fonction Fact utilise la formule de Stirling pour calculer les factorielles des entiers supérieurs à 12. C'est pourquoi le résultat est un nombre flottant.

Syntaxe
p = Fact(nombre)

Le résultat est un nombre flottant, l'argument un nombre entier.

Exemple

p = Fact(52)

résultat : p = 8.06566 e+67

Voir aussi
opérateur ! (factorielle)
Anp
Cnp

Exp (exponentielle)

Exp(nombre)

La fonction **Exp** calcule l'exponentielle d'un nombre.

Syntaxe
Exp(expression)

Le résultat est de type nombre flottant, l'argument de type entier ou flottant

Exemple
Exp(1) est égal à 2.718281828

Voir aussi
Log

Hasard

Hasard(nombre)

Cette fonction permet de donner un nombre entier aléatoire, entre 1 et un maximum donné donné par l'argument de la fonction.

Syntaxe
Hasard(expression)
expression est un nombre entier, le résultat est aussi un nombre entier

Exemple
Hasard(26) donne un nombre entier aléatoire entre 1 et 26.

Log (logarithme)

Log(nombre)

La fonction **Log** calcule le logarithme népérien d'un nombre.

Syntaxe
Log(expression)

Le résultat est de type nombre flottant, l'argument de type entier ou flottant

Exemple
Log(2.718281828) est égal à 1.0

Voir aussi
Exp

Min (plus petit)

Min(...série d'arguments...)

Cette fonction sélectionne le plus petit argument parmi une liste d'arguments, dont le nombre peut être quelconque. Les arguments peuvent être des entiers, ou des listes d'entiers, ou un mélange des deux. Ou bien être des chaînes de caractères, des listes de chaînes, ou un mélange des deux. Dans le cas des arguments listes, l'ensemble des éléments de la liste participe au calcul du minimum.

La fonction **Min** s'adapte aux type de ses arguments : le plus petit nombre est choisi si les arguments sont des nombres entiers ou décimaux, la première chaîne de caractères dans l'ordre alphabétique est choisie si les arguments sont des chaînes de caractères.

Syntaxe
Min(a,b,c...,x)
les arguments sont de type quelconque. la valeur de retour est du même type que l'argument sélectionné pour être la valeur retour.

Exemple sans listes
x = Min(7,3,2,9)
résultat : x = 2

y = Min("Jeux","Aide","Aux","Lettres","De")
résultat : y = "Aide"

Exemple avec listes
L = {5,2,11,3}
x = Min(7,L,9)
résultat : x = 2

Voir aussi
Max

Max (plus grand)

Max(...série d'arguments...)

Cette fonction sélectionne le plus grand argument parmi une liste d'arguments, dont le nombre peut être quelconque. Les arguments peuvent être des entiers, ou des listes d'entiers, ou un mélange des deux. Ou bien être des chaînes de caractères, des listes de chaînes, ou un mélange des deux. Dans le cas des arguments listes, l'ensemble des éléments de la liste participe au calcul du minimum.

La fonction **Max** s'adapte aux type de ses arguments : le plus grand nombre est choisi si les arguments

sont des nombres entiers ou décimaux, la dernière chaîne de caractères dans l'ordre alphabétique est choisie si les arguments sont des chaînes de caractères.

Syntaxe
Max(a,b,c...,x)
les arguments sont de type quelconque. la valeur de retour est du même type que l'argument sélectionné pour être la valeur retour.

Exemple sans listes
x = Max(7,3,2,9)
résultat : x = 9

y = Max("Jeux","Aide","Aux","Lettres","De")
résultat : y = "Lettres"

Exemple avec listes
L = {5,2,11,3}
x = Max(7,L,9)
résultat : x = 11
Voir aussi
Min

Racine

Racine(nombre)

La fonction **Racine** calcule la racine carrée d'un nombre.

Syntaxe
Racine(expression)

Le résultat est de type nombre flottant, l'argument de type entier ou flottant

Exemple
Racine(2) est égal à 1.414214

Voir aussi
opérateur ^ (puissance)

Somme

Somme(...série d'arguments...)

Cette fonction additionne entre eux la liste d'arguments, dont le nombre peut être quelconque. La fonction s'adapte aux type de ses arguments : le résultat est un nombre si les arguments sont des nombres entiers ou décimaux, ou une concaténation de chaîne si les arguments sont des chaînes de caractères. Cette fonction remplace avantageusement une somme explicite utilisant l'opérateur + quand le nombre d'arguments à sommer est variable. Cf l'exemple proposé.

Syntaxe
Somme(a,b,c...,x)
les arguments sont de type quelconque. La valeur de retour est numérique si tous les arguments le sont, sinon c'est une chaîne de caractères

Exemple
//compter le nombre de voyelles A E I O U Y du mot

c = Somme(Comptelettres("A","E","I","O","U","Y",mot))

Opérateur + (addition)

Opérateur +

L'opérateur **+** réalise l'addition numérique entre deux nombres, entiers ou décimaux.

Syntaxe
nbre1 **+** nbre2
Le résultat est de type entier si les deux arguments sont entiers, et de type décimal si l'un des deux arguments est décimal.

Exemple
est-ce vraiment utile ?

Voir aussi
Opérateur + (concaténation)

Opérateur ++ (post incrément)

Opérateur ++ (post incrément)

L'opérateur **++** utilisé en opérateur unaire à gauche, comme dans : nbre**++**
1) renvoie la valeur en cours de l'entier nbre
2) ajoute ENSUITE 1 à l'entier nbre (d'où le nom de post incrément)

Syntaxe
nbre **++**
Le résultat est de type entier.

Exemple

```
a = 10
b = a++
Message(b)
Message(a)
```

La première fonction Message affichera la valeur de b : 10, la même que a. En vertu de la propriété 1) évoquée ci-dessus.
La seconde fonction Message affichera la valeur de a : 11. En vertu de la propriété 2) évoquée ci-dessus.

Voir aussi
Opérateur ++ (pré incrément)

Opérateur ++ (pré incrément)

Opérateur ++ (pré incrément)

L'opérateur **++** utilisé en opérateur unaire à droite, comme dans : **++nbre**
1) ajoute IMMEDIATEMENT 1 à l'entier nbre (d'où le nom de pré incrément)
2) renvoie la nouvelle valeur de nbre.

Syntaxe
++nbre
Le résultat est de type entier.

Exemple

a = 10
b = ++a
Message(b)
Message(a)

Les deux fonctions Message afficheront la valeur 11.

Voir aussi
Opérateur ++ (post incrément)

Opérateur -- (post décrément)

Opérateur -- (post décrément)

L'opérateur **--** utilisé en opérateur unaire à gauche, comme dans : nbre**--**
1) renvoie la valeur en cours de l'entier nbre
2) retire ENSUITE 1 à l'entier nbre (d'où le nom de post décrément)

Syntaxe
nbre **--**
Le résultat est de type entier.

Exemple

a = 10
b = a--
Message(b)
Message(a)

La première fonction Message affichera la valeur de b : 10, la même que a. En vertu de la propriété 1) évoquée ci-dessus.
La seconde fonction Message affichera la valeur de a : 9. En vertu de la propriété 2) évoquée ci-dessus.

Voir aussi
Opérateur -- (pré décrément)

Opérateur -- (pré décrément)

Opérateur -- (pré décrément)

L'opérateur **--** utilisé en opérateur unaire à droite, comme dans : **--nbre**
1) retiree IMMEDIATEMENT 1 à l'entier nbre (d'où le nom de pré incrément)
2) renvoie la nouvelle valeur de nbre.

Syntaxe

--nbre
Le résultat est de type entier.

Exemple

a = 10
b = --a
Message(b)
Message(a)

Les deux fonctions Message afficheront la valeur 9.

Voir aussi
Opérateur -- (post décrément)

Opérateur - (soustraction)

Opérateur -

L'opérateur **-** réalise le soustraction numérique entre deux nombres, entiers ou décimaux.

Syntaxe
nbre1 **-** nbre2
Le résultat est de type entier si les deux arguments sont entiers, et de type décimal si l'un des deux arguments est décimal.

Exemple
est-ce vraiment utile ?

Voir aussi
Opérateur - (éliminer)

Opérateur - (opposé)

Opérateur -

L'opérateur **-** transforme un nombre en son opposé. C'est un des rares opérateurs unaires, c'est à dire qu'il ne possède qu'un seul argument.

Syntaxe
- x

L'argument est entier ou décimal, le résultat de même type

Exemple
x = **- 5**
le résultat est le nombre négatif -5

voir aussi
opérateur - (inverser)

Opérateur * (multiplication)

Opérateur *

L'opérateur ***** réalise la multiplication numérique de deux nombres, entiers ou décimaux.

Syntaxe
nbre1 ***** nbre2
Le résultat est de type entier si les deux arguments sont entiers, et de type décimal si l'un des deux arguments est décimal.

Exemple
est-ce vraiment utile ?

Voir aussi
Opérateur * (multiplication de chaînes)

Opérateur / (division)

Opérateur /

L'opérateur **/** réalise la multiplication numérique de deux nombres, entiers ou décimaux.

Syntaxe
nbre1 **/** nbre2
Le résultat est de type entier si les deux arguments sont entiers, et de type décimal si l'un des deux arguments est décimal.

Exemples
x = 5 / 2
résultat : x = 2
en effet, 5 et 2 sont entiers, et le résultat est donc traité comme un nombre entier

x = 5.0 / 2
résultat : x = 2.5
en effet, l'un des deux arguments (5.0) est décimal, le résultat est donc traité comme un nombre décimal.

Opérateur % (modulo)

Opérateur %

L'opérateur **%** calcule la valeur de l'argument de gauche , modulo l'argument de droite. Formellement, le modulo est le reste de la division entière de l'argument de gauche divisé par l'argument de droite.

Syntaxe
nbre1 **%** nbre2
Le résultat est de type entier et les deux arguments sont considérés comme entiers

Exemples
x = 11 % 3
résultat : x = 2
en effet, le reste de la division entière de 11 par 3 est 2 (11 = 3*3 + 2)

x = 8 % 4
résultat : x = 0
en effet, le reste de la division entière de 8 par 4 est 0 (8 = 2*4 + 0)

Opérateur ^ (puissance)

Opérateur ^

L'opérateur **^** calcule la valeur de l'argument de gauche élevé à la puissance de l'argument de droite. Les particularités suivantes s'appliquent, comme il se doit selon les règles des mathématiques :

- Si la puissance est entière et l'argument de gauche est négatif, le calcul est accepté, et la valeur est négative si l'argument puissance est impair.
- Si la puissance est entière et l'argument de gauche est négatif, le calcul est accepté, et la valeur est positive si l'argument puissance est pair.
- Si l'argument de gauche est négatif, mais l'argument de droite non-entier, le calcul est refusé.

Syntaxe
nbre1 **^** nbre2
Le résultat est de type entier si les deux arguments sont entiers et la puissance positive. De type décimal sinon.

Exemples
```
a = 2 ^ 2
a = -2 ^ 2
a = 2 ^ -2
a = -2 ^ -2

a = 3 ^ 3
a = -3 ^ 3
a = 3 ^ -3
a = -3 ^ -3

a = 2 ^ 2.0
//a = -2 ^ 2.0 est interdit
a = 2 ^ -2.0
//a = -2 ^ -2.0 est interdit

a = 3.0 ^ 3
a = -3.0 ^ 3
a = 3.0 ^ -3
a = -3.0 ^ -3
```

donne ceci (mode débug activé pour générer une trace automatique de tous les calculs)
[Debug] a = 4
[Debug] a = 4
[Debug] a = 0.250000
[Debug] a = 0.250000

[Debug] a = 27
[Debug] a = -27
[Debug] a = 0.037037
[Debug] a = -0.037037

[Debug] a = 4.000000
[Debug] a = 0.250000

[Debug] a = 27.000000
[Debug] a = -27.000000
[Debug] a = 0.037037
[Debug] a = -0.037037

Opérateur && (et logique)

Opérateur &&

Synonyme : Et

Cet opérateur sur opérandes booléens répond à la logique du **ET logique.**
C'est à dire que le seul cas où le résultat est VRAI est celui où les deux opérandes sont VRAI. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

table de vérité
VRAI && VRAI = VRAI
VRAI && FAUX = FAUX
FAUX && VRAI = FAUX
FAUX && FAUX = FAUX

Syntaxe
expression && expression
expression Et expression

Exemple
```
si( a > 0 && a < 10)
```

le booléen évalué dans les parenthèses de cette instruction si() est VRAI si et seulement si le nombre a est strictement supérieur à 0 et strictement inférieur à 10.

Voir aussi
opérateur ou
opérateur non

Opérateur || (ou logique)

Opérateur ||

Synonyme : Ou

Cet opérateur sur opérandes booléens répond à la logique du **OU logique**.
C'est à dire que le seul cas où le résultat est FAUX est celui où les deux opérandes sont FAUX. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

table de vérité
VRAI || VRAI = VRAI
VRAI || FAUX = VRAI
FAUX || VRAI = VRAI
FAUX || FAUX = FAUX

Syntaxe
expression || expression
expression Ou expression

Exemple
```
si( a <= 0 || a >= 10)
```

le booléen évalué dans les parenthèses de cette instruction si() est VRAI si et seulement si le nombre a est inférieur à 0 ou supérieur à 10.

Voir aussi
opérateur et
opérateur non

Opérateur ! (non logique)

Opérateur !

Cet opérateur sur un opérande booléen placé à sa droite répond à la logique du **NON logique**.
Il transforme simplement le booléen VRAI en FAUX, et réciproquement. C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé immédiatement après l'opérateur..

table de vérité
! VRAI = FAUX
! FAUX = VRAI

Syntaxe
! expression

Exemple
```
si( !(a > 10) )
```

le booléen évalué dans les parenthèses de cette instruction si() est VRAI si et seulement si le nombre a n'est pas strictement supérieur à 10.

Voir aussi
opérateur et
opérateur ou

Opérateur < (inférieur)

Opérateur <

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est strictement plus petit que l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 < argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX.

Exemples

```
1 < 10  est VRAI
1 < 1 est FAUX
"ABC" < "ABCDEF" est VRAI
"ABC" < "ABB" est FAUX
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur <= (inférieur ou égal)

Opérateur <=

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est plus petit ou égal que l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 <= argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX.

Exemples

```
1 <= 10  est VRAI
1 <= 1 est VRAI
"ABC" <= "ABCDEF" est VRAI
"ABC" <= "ABB" est FAUX
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur == (égal)

Opérateur ==

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est égal à l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 == argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX

Exemples
```
1 == 10  est FAUX
1 == 1 est VRAI
"ABC" == "abc" est FAUX
"ABC" == "ABC" est VRAI
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur != (différent)

Opérateur !=

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est différent de l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 != argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX

Exemples
```
1 != 10  est VRAI
1 != 1 est FAUX
"ABC" != "abc" est VRAI
"ABC" != "ABC" est FAUX
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=

opérateur >=
opérateur >

Opérateur >= (supérieur ou égal)

Opérateur >=

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est plus grand ou égal que l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 >= argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX.

Exemples
```
1 >= 10  est FAUX
1 >= 1 est VRAI
"ABC" >= "ABCDEF" est FAUX
"ABC" >= "ABB" est VRAI
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur > (supérieur)

Opérateur >

Cet opérateur compare deux arguments et donne le résultat booléen VRAI si l'argument de gauche est plus grand strictement que l'argument de droite. Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

Si aucun opérande n'est une chaîne de caractères, la comparaison est réalisée selon l'ordre numérique des arguments entiers ou décimaux.
Si les deux arguments sont des chaînes de caractères, la comparaison est réalisée selon l'ordre alphabétique.
Si un seul argument est une chaîne de caractères et l'autre n'est pas une chaîne de caractères, la comparaison est rejetée en erreur.

Syntaxe
argument 1 > argument2
les arguments sont entiers, décimaux ou chaîne de caractères. le résultat et un booléen VRAI ou FAUX.

Exemples

```
1 > 10  est FAUX
1 > 1 est FAUX
"ABCZ" > "ABCDEF" est VRAI
"ABC" > "ABC" est FAUX
```

Voir aussi

opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur & (et binaire)

Opérateur &

Cet opérateur sur opérandes entier réalise le **ET binaire**, bit à bit; entre 2 nombres. Compte tenu de son utilisation naturelle, le résultat est automatiquement traité en hexadécimal

table de vérité binaire
1 & 1 = 1
1 & 0 = 0
0 &1 = 0
0 & 0 = 0

Opérateur | (ou binaire)

Opérateur |

Cet opérateur sur opérandes entier réalise le **OU binaire**, bit à bit; entre 2 nombres. Compte tenu de son utilisation naturelle, le résultat est automatiquement traité en hexadécimal.

table de vérité binaire
1 | 1 = 1
1 | 0 = 1
0 | 1 = 1
0 | 0 = 0

Opérateur &| (ou exclusif)

Opérateur &|

Cet opérateur sur opérandes entier réalise le **OU EXCLUSIF binaire**, bit à bit; entre 2 nombres. Compte tenu de son utilisation naturelle, le résultat est automatiquement traité en hexadécimal.

table de vérité binaire
1 &| 1 = 0
1 &| 0 = 1
0 &| 1 = 1
0 &| 0 = 0

Opérateur ! (factorielle)

Opérateur !

L'opérateur **n!** calcule la factorielle de n. C'est la notation usuelle en mathématique, qui calcule le nombre de **permutations** sans répétition de n objets entre eux.C'est un opérateur unaire à gauche, c'est à dire qu'il ne possède qu'un seul argument placé immédiatement avant l'opérateur.

Syntaxe
p = nombre!

Le résultat est un nombre entier, l'argument aussi.

Exemple

p = 4!

résultat : p =24. Il y a en effet 24 permutations possibles de 4 objets a b c d

abcd abdc acbd acdb adbc adcb

bacd badc bcad bcda bdac bdca

cadb cabd cbad cbda cdab cdba

dabc dacb dbac dbca dcab dcba

Nota

La fonction factorielle produit très vite des nombres prodigieusement grands. Ainsi, le nombre de tirages différents d'un jeu de cartes de 52 cartes, c'est à dire 52!, est un nombre de 68 chiffres. C'est un nombre plus grand que le nombre d'atomes dans tout l'univers.

La plus grande factorielle calculable par l'opérateur ! est 12! = 479.001.600

La fonction Fact, équivalente à l'opérateur ! jusqu'à 12!, n'a pas cette limitation car le résultat produit est un nombre flottant.

Voir aussi

Anp
Cnp
fonction Fact

Listes

Fonctions opérant sur listes

voir une présentation globale du concept de liste dans cet article

Assemble	rassemble les éléments d'une liste sous forme d'une chaîne de caractères
Change	modifie un élément à une certaine position d'une liste
Dicoliste	crée une liste avec les mots du dictionnaire
Dissocie	range les éléments d'une liste dans un tableau
Insere	insère un ou plusieurs éléments à une certaine position d'une liste
Inter	intersection de deux listes
Lireliste	instruction-bloc de lecture itérative d'une liste
Max	élément maximum d'une ou plusieurs listes
Min	élément minimum d'une ou plusieurs listes
Partie	sélectionne un nombre d'éléments consécutifs d'une liste
Retire	retire certains éléments d'une liste
Separe	sépare une chaîne de caractères en éléments d'une liste
Union	union de deux listes

Opérateur {} (liste)	construit une liste en spécifiant explicitement ses éléments
Opérateur % (série)	construit une liste de nombres consécutifs, commençant par 1
Opérateur % (taille)	nombre d'éléments d'une liste
Opérateur $ (caractères)	construit une liste avec chaque caractère d'une chaîne
Opérateur $ (chaîne)	rassemble les éléments d'une liste dans une seule chaîne de caractères
Opérateur + (ajout)	ajoute les éléments d'une liste à la suite de ceux d'une autre liste
Opérateur - (élimine)	élimine d'une liste tous les éléments présents dans une autre liste
Opérateur - (inverse)	inverse l'ordre des éléments d'une liste
Opérateur * (multiplie)	réplique une liste, ou chaque élément d'une liste
Opérateur :: (index)	sélectionne les éléments d'une liste en fonction de leur index
Opérateur ^ (premier)	sélectionne le premier élément d'une liste
Opérateur ? (recherche)	recherche les éléments d'une liste dans une autre liste
Opérateur & (masque)	sélectionne les éléments d'une liste selon un masque
Opérateur µ (aplatit)	ramène au 1er niveau les éléments de listes imbriqués
Opérateur µ (regroupe)	regroupe par paquets les éléments d'une liste
Opérateur %< (tri)	trie une liste numérique dans l'ordre ascendant
Opérateur %> (tri)	trie une liste numérique dans l'ordre descendant
Opérateur $< (tri)	trie une liste de chaînes dans l'ordre ascendant
Opérateur $> (tri)	trie une liste de chaînes dans l'ordre descendant
Opérateur <> (rotation)	effectue un décalage à droite ou à gauche, les éléments éjectés d'un côté réapparaissent de l'autre
Opérateur << (décale)	effectue un décalage à gauche, les éléments éjectés sont supprimés.
Opérateur >> (décale)	effectue un décalage à droite, les éléments éjectés sont supprimés.

Opérateur <	comparaison de listes : opérateur "inférieur"
Opérateur <=	comparaison de listes : opérateur "inférieur ou égal"
Opérateur ==	comparaison de listes : opérateur "égal"
Opérateur !=	comparaison de listes : opérateur "différent"
Opérateur >=	comparaison de listes : opérateur "supérieur ou égal"
Opérateur >	comparaison de listes : opérateur "supérieur"

°distribution	distribue un opérateur unaire ou binaire sur les éléments d'une ou plusieurs listes, pour obtenir une liste

réduction° applique un opérateur binaire sur les éléments d'une seule liste, pour obtenir un seul élément
°produit externe° distribue un opérateur binaire sur chaque couple possible des éléments de deux listes, pour obtenir une liste

Assemble

Assemble(Liste,separateur)

Cette fonction assemble les éléments d'une liste sous forme d'un unique chaîne de caractères, avec ajout d'un séparateur entre chaque élément

Syntaxe
`Assemble(LISTE,sep)`
Le résultat est un chaîne rassemblant les éléments de **`LISTE`** séparés chacun par le séparateur **`sep`** choisi.

Exemple
```
LISTE = {"tigre","lion","escargot"}
item = Assemble(LISTE," suivi d'un ")
```
Le résultat est la chaîne de caractères `"tigre suivi d'un lion suivi d'un escargot"`

Voir aussi
Separe

Change (liste)

Change(Liste,index,valeur)

Cette fonction remplace un ou plusieurs éléments d'une liste. Les éléments à remplacer sont précisés par l'argument index, qui peut être soit un entier simple, soit une liste d'entiers, correspondant aux rangs (à partir de 1) des éléments concernés. Les éléments de remplacement sont spécifiés dans le 3ème argument valeur, dans le même ordre que les index.Le premier argument **Liste** est toujours une liste, le second **index** est soit un nombre entier simple, soit une liste. Le dernier **valeur** est soit un élément simple, soit une liste de même taille qu'**index.**

Les index inférieurs strictement à 1 ou supérieurs strictement au nombre d'éléments de Liste sont simplement ignorés. Une erreur est générée sur les listes **index** et **valeur** ne sont pas de même taille.

Syntaxe
`Change(LISTE,index,valeur)`
Le résultat est la liste **`LISTE`** dont on a remplacé l'élément spécifié par **`index`** par le remplacement spécifié dans **`valeur`**

`Change(LISTE,INDEX,VALEURS)`
Le résultat est la liste **`LISTE`** dont on a remplacé les éléments spécifiés par **`INDEX`** par les remplacements spécifiés dans **`VALEURS`**

`Change(LISTE,INDEX,valeur)`
Le résultat est la liste **`LISTE`** dont on a remplacé les éléments spécifiés par **`index`** par le remplacement unique spécifié dans **`valeur`**

Exemples

```
LISTE = {"tigre","lion","escargot"}
LISTE = Change(LISTE,3,"puma")
```

Le résultat est la liste `{"tigre","lion","puma"}`

```
LISTE = {"tigre","lion","léopard","limace"}
//attention, les deux listes (index et nouvelles valeurs) doivent être de même taille
LISTE = Change(LISTE,{4,1,8},{"guépard","puma","dauphin"})
```

Le résultat est la liste `{"puma","lion","léopard","guépard"}`

```
LISTE = {"tigre","lion","léopard","limace"}
LISTE = Change(LISTE,{1,3},"dauphin")
```

Le résultat est la liste `{"dauphin","lion","dauphin","limace"}`

Voir aussi

Insere
Retire
Change (opérant sur chaîne de caractère)

Dissocie (liste)

Dissocie(nom,LISTE)

La fonction **Dissocie** collecte un à un les éléments de la liste en second argument, et les range dans des éléments de tableau, dont le nom est précisé par le premier argument. Le premier élément est rangé dans l'indice 1. Le résultat de la fonction est un nombre entier égal à la taille de la liste, et donc aussi égal au dernier indice de tableau utilisé.

Syntaxe

Dissocie(nom,liste)
Le résultat est de type entier.

Exemple

```
liste = {"AZE",{1,2,3},99}
x = Dissocie(Tableau,liste)
```

Le résultat est 3 (taille de liste). Par ailleurs, les 3 éléments de tableau suivants ont été affectés :
Tableau[1] = "AZE"
Tableau[2] = {1,2,3}
Tableau[3] = 99

Cette fonction est particulièrement utile pour initialiser d'un coup plusieurs éléments d'un tableau :

```
n = Dissocie(Code,{"alpha","bravo","charlie","delta","echo","foxtrot"})
```

est plus élégant que :

```
Code[1] = "alpha"
Code[2] = "bravo"
Code[3] = "charlie"
Code[4] = "delta"
Code[5] = "echo"
Code[6] = "foxtrot"
```

Insere (liste)

Insere(Liste,index,valeur)

Cette fonction insère des éléments dans une liste, à partir d'un index spécifié. Le premier élément inséré prend place directement après l'index spécifié. Les index suivants sont occupés par la suite des valeurs insérées. Les valeurs précédentes de la Liste, qui occupaient ces index, sont simplement décalées à la suite de la dernière valeur insérée. Le premier argument **Liste** est toujours une liste, le second **index** est toujours un nombre entier simple, le dernier **valeur** peut être un élément simple ou une liste.

Fonctionnement détaillé :

- Un index égal à 0 provoque une insertion en tout début de liste, l'ensemble des éléments de la liste est décalé à droite.
  - Insere({1,2,3},0,"x"} --> {"x",1,2,3}
- Un index supérieur à 0 mais inférieur ou égal au nombre d'éléments provoque une insertion juste après le rang spécifié par l'index. Notamment un index égal au nombre d'éléments insère la ou les nouvelles valeurs juste après le dernier élément de liste.
  - Insere({1,2,3},1,"x"} --> {1,"x",2,3}
  - Insere({1,2,3},2,"x"} --> {1,2,"x",3}
  - Insere({1,2,3},3,"x"} --> {1,2,3,"x"}

Syntaxe
`Insere(LISTE,index,valeur)`
Le résultat est la liste **`LISTE`** dont les éléments spécifiés dans **`valeur`** viennent s'insérer juste après le rang **`index.`**

Exemples
```
LISTE = {"tigre","lion"}
LISTE = Insere(LISTE,0,"escargot")
```
Le résultat est la liste `{"escargot","tigre","lion"}`

```
LISTE = {"tigre","lion","léopard"}
LISTE = Insere(LISTE,1,{"limace","dauphin"})
```
Le résultat est la liste `{"tigre","limace","dauphin","lion","léopard"}`

Voir aussi
Retire
Change

Inter

Inter(Liste1,Liste2)

La fonction Inter détermine l'intersection des éléments de deux listes triées en ordre croissant, c'est à dire l'ensemble des éléments communs aux deux listes.
L'algorithme utilisé est ultra-rapide, donc adapté au traitement des très grandes listes, mais il exige que les deux listes soient triées dans un ordre croissant.

Cas des doublons
Si des éléments sont en double dans l'une ou l'autre des listes, un seul des exemplaires est conservé dans la liste résultat.

Syntaxe
`Inter(L1,L2)`
Le résultat est l'ensemble des éléments présents à la fois dans L1 et L2. C'est aussi une liste triée dans l'ordre croissant.

Exemple
```
L1 ={"limace","lion","lion","tigre"}
L2 =  {"dauphin","escargot","limace","limace","lion"}
LISTE = Inter(L1,L2)
```

Le résultat est la liste `{"limace","lion"}`

Voir aussi
Union

Max

Max(...série d'arguments...)

Cette fonction sélectionne le plus grand argument parmi une liste d'arguments, dont le nombre peut être quelconque. Les arguments peuvent être des entiers, ou des listes d'entiers, ou un mélange des deux. Ou bien être des chaînes de caractères, des listes de chaînes, ou un mélange des deux. Dans le cas des arguments listes, l'ensemble des éléments de la liste participe au calcul du minimum. L'ensemble des éléments des éventuelles listes imbriquées est également pris en compte.

La fonction **Max** s'adapte aux type de ses arguments : le plus grand nombre est choisi si les arguments sont des nombres entiers ou décimaux, la dernière chaîne de caractères dans l'ordre alphabétique est choisie si les arguments sont des chaînes de caractères.

Syntaxe
Max(a,b,c...,x)
les arguments sont de type quelconque. la valeur de retour est du même type que l'argument sélectionné pour être la valeur retour.

Exemple sans listes
x = Max(7,3,2,9)
résultat : x = 9

y = Max("Jeux","Aide","Aux","Lettres","De")
résultat : y = "Lettres"

Exemple avec listes
M = {11,2}
L = {5,2,10,M,3}
x = Max(7,L,9)
résultat : x = 11
Voir aussi
Min

Min

Min(...série d'arguments...)

Cette fonction sélectionne le plus petit argument parmi une liste d'arguments, dont le nombre peut être quelconque. Les arguments peuvent être des entiers, ou des listes d'entiers, ou un mélange des deux. Ou bien être des chaînes de caractères, des listes de chaînes, ou un mélange des deux. Dans le cas des arguments listes, l'ensemble des éléments de la liste participe au calcul du minimum. L'ensemble des éléments des éventuelles listes imbriquées est également pris en compte.

La fonction **Min** s'adapte aux type de ses arguments : le plus petit nombre est choisi si les arguments sont des nombres entiers ou décimaux, la première chaîne de caractères dans l'ordre alphabétique est choisie si les arguments sont des chaînes de caractères.

Syntaxe
Min(a,b,c...,x)
les arguments sont de type quelconque. la valeur de retour est du même type que l'argument sélectionné pour être la valeur retour.

Exemple sans listes
x = Min(7,3,2,9)
résultat : x = 2

y = Min("Jeux","Aide","Aux","Lettres","De")
résultat : y = "Aide"

Exemple avec listes
M = {11,2}
L = {5,4,10,M,3}
x = Min(7,L,9)
résultat : x = 2

Voir aussi
Max

Partie (liste)

Partie(Liste,index,nombre)

Cette fonction sélectionne un nombre donné d'éléments dans une liste, à partir d'un index spécifié. Le premier élément sélectionné est celui dont l'index est spécifié par le deuxième argument. Le troisième argument indique le nombre d'éléments total à sélectionner, à partir de l'index donné.
Les valeurs illégales (négatives, en débordement, etc...) résultant des choix de valeurs **`index`** et **`nombre`** sont simplement ignorées

Syntaxe
`Partie(LISTE,index,nombre)`
Le résultat est une liste de **`nombre`** éléments de la liste **`LISTE,`** sélectionnés à partir du rang **`index.`**

Exemples
LISTE = `{"tigre","limace","dauphin","lion","léopard"}`
`PARTIE = Partie(LISTE,2,3)`

Le résultat est la liste `{"limace","dauphin","lion"}`

Retire (liste)

Retire(Liste,index)

Cette fonction supprime un ou plusieurs éléments d'une liste. Les éléments à supprimer sont précisés par l'argument index, qui peut être soit un entier simple, soit une liste d'entiers, correspondant aux rangs (à partir de 1) des éléments concernés.

Les index inférieurs strictement à 1 ou supérieurs strictement au nombre d'éléments de Liste sont simplement ignorés.

Syntaxe
`Retire(LISTE,index)`
Le résultat est la liste **`LISTE`** dont on a retiré les éléments spécifiés par **`index.`**

Exemples
```
LISTE = {"tigre","lion","escargot"}
LISTE = Retire(LISTE,3)
```
Le résultat est la liste `{"tigre","lion"}`

```
LISTE = {"tigre","lion","escargot","puma","léopard","limace","guépard"}
LISTE = Retire(LISTE,{3,6})
```
Le résultat est la liste `{"tigre","lion","puma","léopard","guépard"}`

Voir aussi
Insere
Change

Separe

Separe(chaine,separateurs)

Cette fonction découpe une chaîne de caractères sous forme d'éléments d'une liste. Pour celà on fournit une chaîne de caractères regroupant les caractères séparateurs autorisés à délimiter chaque élément.

Syntaxe
`Separe(chaine,sep)`
Le résultat est une liste des caractères de **`chaine`** présents entre un ou plusieurs des caractères séparateurs spécifiés dans la chaîne de caractères **`sep`**

Exemple
```
chaine = "tigre/lion//puma/+/escargot+limace"
LISTE = Separe(chaine,"/+")
```
Le résultat est la liste `{"tigre","lion","puma","escargot","limace"}`

Voir aussi
Assemble

Union

Union(Liste1,Liste2)

La fonction Union détermine la réunion des éléments de deux listes triées en ordre croissant, c'est à dire l'ensemble des éléments présents dans les deux listes.
L'algorithme utilisé est ultra-rapide, donc adapté au traitement des très grandes listes, mais il exige que les deux listes soient triées dans un ordre croissant.
Cas des doublons
Si des éléments sont en double dans l'une ou l'autre des listes, un seul des exemplaires est conservé dans la liste résultat.

Syntaxe
`Union(L1,L2)`
Le résultat est l'ensemble des éléments présents dans L1 ou L2. C'est aussi une liste triée dans l'ordre croissant.

Exemple
```
L1 ={"limace","lion","tigre","tigre"}
L2 =   {"dauphin","escargot","limace","limace","lion"}
LISTE = Union(L1,L2)
```

Le résultat est la liste `{"dauphin","escargot","limace","lion","tigre"}`

Voir aussi
Inter

Opérateur {} (liste)

Opérateur { }

L'opérateur { } crée une liste à partir des éléments donnés en arguments, entre les accolades, séparés par des virgules. Une liste vide est crée par l'expression { }

Syntaxe
`{x,y,...}`
Le résultat est une liste des éléments entre accolades

Exemples
```
LISTE = {"truc","machin","bidule"}
LISTE = {1,2}

liste contenant deux listes :
LISTE = { {1,2,3}, {"a","b","c"} }

liste vide
VIDE = {}
```

Opérateur % (série)

Opérateur %

L'opérateur `%` crée une liste de nombres commençant par 1, et se terminant par le nombre donné en argument. C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite. Cas particulier : si l'argument est inférieur ou égal à zéro, la liste obtenue est une liste vide.

Syntaxe
`%n`
Le résultat est une liste de nombres {1,...etc..., **`n`**}

Exemples
```
item = %10
```
le résultat est la liste {1,2,3,4,5,6,7,8,9,10}

```
item = %0
```
le résultat est la liste vide { }

Voir aussi
opérateur % (nombre d'éléments)

Opérateur % (nombre)

Opérateur %

L'opérateur `%` donne le nombre d'éléments d'une liste. C'est un opérateur unaire à gauche, c'est à dire qu'il ne possède qu'un seul argument placé à sa gauche

Syntaxe
`LISTE%`
Le résultat est le nombre d'éléments de la liste **`LISTE`**

Exemples
```
LISTE = {"vis","marteau","vis","escargot","scie"}
item = LISTE%
```
le résultat est 5

```
LISTE = { {1,2,3}, {"a","b","c"} }
item = LISTE%
```
le résultat est 2

Voir aussi
opérateur % (série de nombres)

Opérateur $ (caractères)

Opérateur $

L'opérateur **`$`** crée une liste avec chacun des caractères de la chaîne de caractères donnée en argument.

C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
`$chaine`
Le résultat est une liste des caractères de **`chaine`**

Exemples
```
item = $"Azerty"
```
le résultat est la liste {"A","z","e","r",",t","y"}

Voir aussi
opérateur $ (regroupement en une chaîne)

Opérateur $ (chaîne)

Opérateur $

L'opérateur $ regroupe l'ensemble des éléments d'une liste, et ses éventuelles sous-listes, sous forme d'une unique chaîne de caractères. C'est un opérateur unaire à gauche, c'est à dire qu'il ne possède qu'un seul argument placé à sa gauche

Syntaxe
`LISTE$`
Le résultat est une chaîne de caractères regroupant les éléments de **`LISTE`**

Exemples
```
LISTE = {"vis",142857,"escargot"}
item = LISTE$
```
le résultat est "vis142857escargot"

```
LISTE = { {1,2,3}, {"a","b","c"} }
item = LISTE$
```
le résultat est "123abc"

Voir aussi
opérateur $ (liste de caractères)

Opérateur + (ajoute)

Opérateur +

L'opérateur **+** ajoute les éléments de la liste de droite à la suite de ceux de la liste de gauche. L'argument de droite peut-être indifféremment une liste ou un élément simple.

Syntaxe
`LISTE + AJOUT`
Le résultat est la liste des éléments de **`LISTE`** complétés de ceux de **`AJOUT`**

`LISTE + x`
Le résultat est la liste des éléments de **`LISTE`** complétée de **`x`**

Exemples

```
LISTE = {2,4,6}
item = LISTE + {3,5,7}
```
le résultat est la liste {2,4,6,3,5,7}

```
LISTE = {2,4,6}
item = LISTE + 7
```
le résultat est la liste {2,4,6,7}

Voir aussi
opérateur + (addition de chaîne de caractères)

Opérateur - (inverse)

Opérateur -

L'opérateur **–** inverse les éléments d'une liste. C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
-LISTE
Le résultat la liste des éléments de **LISTE** dans l'ordre inverse.

Exemples

```
LISTE = {2,4,7,1,3,5}
item = - LISTE
```
le résultat est la liste {5,3,1,7,4,2}

Voir aussi
opérateur - (inversion de chaîne de caractères)
opérateur - (élimination d'éléments de liste)

Opérateur - (élimine)

Opérateur -

L'opérateur **–** élimine de la liste de gauche, tous les éléments présents dans la liste de droite. L'argument de droite peut-être indifféremment une liste ou un élément simple.

Syntaxe
LISTE - RETIRE
Le résultat est la liste des éléments de **LISTE** de laquelle on a ôté ceux présents (en quelque nombre que ce soit) dans la liste **RETIRE**

LISTE - x
Le résultat est la liste des éléments de **LISTE** de laquelle on a ôté tous ceux égaux à **x**

Exemples

```
LISTE = {1,2,3,4,4,5,5,6,7,8,9,9}
item = LISTE - {1,3,5,7,9,11,13}
```
le résultat est la liste {2,4,4,6,8}

```
LISTE = {1,2,4,5,6,5,8,9,5}
item = LISTE - 5
```
le résultat est la liste `{1,2,4,6,8,9}`

Voir aussi
opérateur - (élimination de chaîne de caractères)
opérateur - (inversion de liste)

Opérateur * (multiplie)

Opérateur *

L'opérateur ***** multiplie toute une liste, ou chaque élément d'une liste, le nombre de fois spécifié par le 2ème argument de la "multiplication". Le résultat est différent selon que l'opérande liste est à droite ou à gauche

Syntaxe 1
`LISTE * m`
Le résultat est **`m`** fois la liste des éléments de **`LISTE`**

Syntaxe 2
`m * LISTE`
Le résultat est une liste des éléments de **`LISTE`** multipliés chacun **`m`** fois

Exemples

```
LISTE = {2,4,6}
item = LISTE * 3
```
le résultat est la liste {2,4,6,2,4,6,2,4,6}

```
item = 3 * LISTE
```
le résultat est la liste {2,2,2,4,4,4,6,6,6}

Voir aussi
opérateur * (multiplication de chaîne de caractères)

Opérateur :: (index)

Opérateur ::

1) Utilisé en partie droite

L'opérateur **::** sélectionne un ou plusieurs éléments d'une liste. On précise en premier argument (gauche) la liste, et en second argument (droite) l'index de l'élément à sélectionner, ou bien une liste d'index. Un index 1 sélectionne le premier élément. Le résultat est un élément simple ou une liste selon que l'index est un élément simple ou une liste.

Syntaxes possibles
1) index est un élément simple

LISTE::index
Le résultat est l'unique élément de liste dont le rang est donné par l'argument index. Si index est < 0 ou plus grand que le nombre d'éléments de la liste, le résultat est une chaîne vide.

2) INDEX est une liste d'index
LISTE::INDEX
Le résultat est une liste des éléments de **LISTE** correspondants aux rangs donnés dans la liste **INDEX**. Si un des index est < 0 ou plus grand que le nombre d'éléments de la liste, une erreur est générée.

Exemples
`LISTE = {"truc","machin","bidule"}`
`item = LISTE::2`
le résultat est l'élément simple "machin"

`item = LISTE::{3}`
le résultat est la liste {"bidule"}

`item = LISTE::{1,3,1}`
le résultat est la liste {"truc","bidule","truc"}

Voir aussi
opérateur :: (index sur chaîne de caractères)
opérateur ? (recherche)

2) Utilisé en partie gauche

L'opérateur **::** permet d'affecter une valeur (une seule) à un élément (un seul) de la liste, désigné par son index. La liste doit contenir un nombre d'éléments au moins égal à l'index utilisé.

Syntaxe
LISTE::index = valeur

Exemple
`LISTE = {"truc","machin","bidule"}`
`LISTE::2 = "escargot"`
la liste `LISTE` contient désormais `{"truc","escargot","bidule"}`

Voir aussi
opérateur :: (index sur chaîne de caractères)
opérateur ? (recherche)

Opérateur ^ (premier)

Opérateur ^

L'opérateur **^** sélectionne le premier élément d'une liste. Il est unaire à droite. Il est strictement équivalent à l'appel à l'opérateur index, avec l'argument 1 à droite.

Syntaxe
^LISTE
Le résultat est l'unique élément de liste de rang 1.

Exemples
`LISTE = {"truc","machin","bidule"}`

```
item = ^LISTE
```
le résultat est l'élément simple "truc"

Opérateur ? (recherche)

Opérateur ?

L'opérateur **?** est un opérateur de recherche dans une liste. Il fonctionne différemment selon que l'on y recherche un ou plusieurs éléments.

Syntaxe 1
`LISTE ? ref`
ref est un élément simple

Si l'opérande de gauche est une liste, et celui de droite un élément simple, l'opérateur recherche les occurrences de l'élément simple dans la liste de gauche. Le résultat est une liste d'index, telle que la valeur des éléments de la liste à ces positions est égale à l'élément simple recherché. Si aucun élément ne correspond, la liste rendue est vide.Le résultat est une liste des index {ix,...} telle que LISTE::ix = ref.

Syntaxe 2
`LISTE ? REF`
REF est une liste.

Si les deux opérandes sont des listes, l'opérateur recherche chaque élément de la liste spécifiée à droite, parmi ceux de la liste à gauche. Les éléments de la liste résultat sont :

- Si l'élément est trouvé, l'élément rendu est son index dans la liste de gauche
- si l'élément est absent, l'élément rendu est le nombre d'éléments de la liste de gauche, augmenté de 1.

Le résultat est une liste d'index dans la liste **`LISTE`**, de même taille que la liste **`REF`**. Le premier index est l'index dans **`LISTE`** du 1er élément de **`LISTE`** égal à l'élément de rang 1 de **`REF`**. Le second index est l'index du 1er élément de **`LISTE`** égal à l'élément de rang 2 de **`REF`**. etc...

Exemples
```
LISTE = {"marteau","tournevis","clou","vis","marteau","clé à molette","pince coupante","scie"}
REF = {"vis","marteau","vis","escargot","scie"}
```

```
item = LISTE ? "marteau"
```
le résultat est {1,5} car les éléments de rang 1 et 5 de `LISTE` sont égaux à "marteau"

```
item = LISTE ? REF
```
le résultat est {4,1,4,9,8}, car le 1er élément de REF "vis" correspond au rang 4 de `LISTE`, le 2ème "marteau" au rang 1, le 3ème, "vis" encore au rang 4, le 4ème "escargot" absent de la liste donne 9, le 5ème "scie" est trouvé au rang 8 de la liste `LISTE`.
Notez que l'index 5 ne sera jamais rendu, pour cause de doublon car "marteau" est déjà en 1. Notez que "escargot" absent de la liste donne un index 9 car `LISTE` a 8 éléments.

Voir aussi
opérateur ? (recherche sur chaîne de caractères)
opérateur :: (index)

Opérateur & (masque)

Opérateur &

L'opérateur **&** sélectionne les éléments d'une liste spécifiée en argument gauche, correspondants aux 1 de même rang dans une autre liste spécifiée en argument droite

Syntaxe
LISTE & MASQUE

Le résultat est une liste des éléments de **LISTE** correspondants aux 1 (valeur numérique) de même rang dans la liste **MASQUE**

Exemples
```
LISTE = {4,2,6,7,"truc",0}
item = LISTE & {0,0,0,1,1,0}
```
le résultat est la liste {7,"truc"}

En pratique, la liste masque est souvent le résultat d'un calcul faisant intervenir des opérateurs booléens distribués sur les éléments de 2 listes que l'on compare entre elles.
```
R1 = {"clou","pince coupante","marteau","tournevis","scie"}
R2 = {"clé à molette","clou","marteau","pince coupante","scie"}
item = R1 & (R1 °== R2)
```
le résultat est la liste {"marteau","scie"}, qui sont les seuls éléments identiques au même rang dans les deux listes, condition exprimée par `R1 °== R2`

Opérateur µ (aplatit)

Opérateur µ

L'opérateur **µ** travaille sur les listes contenant d'autres listes. Il construit une liste d'un seul niveau avec l'ensemble des éléments contenus dans la liste en argument, quelque soit le niveau d'imbrication où ces éléments se trouvent. Les liste vides sont éliminées au passage. C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
µLISTE
Le résultat est une liste aplatie des éléments de **LISTE**

Exemple
```
LISTE = { {"AF","DEF"} , "AZ", {"DG", {"Y","XZ"} }, { } }
L2 = µLISTE
le résultat est {"AF","DEF","AZ","DG","Y","XZ"}
```

Voir aussi
opérateur µ (regroupement)

Opérateur µ (regroupe)

Opérateur µ

L'opérateur **µ** ajoute un niveau d'imbrication à la liste placée en argument gauche, en regroupant ses

éléments par sous-listes de la taille spécifiée par l'argument droite.

Syntaxe
`LISTE µ nbre`
Le résultat est une liste des éléments de **`LISTE`** regroupés par paquets de **`nbre`** éléments.

Exemple
`LISTE = {"AF","DEF",DG","XZ",3,4} µ 2`
Le résultat est { `{"AF","DEF"} , {"DG","XZ"} ,{3,4} }` les 6 éléments de `LISTE` ont été regroupés par paquets de 2, ce qui nécessite 3 sous-listes de 2 éléments chacune.

Voir aussi
[opérateur µ (aplatissement)]

Opérateur %> (tri)

Opérateur %>

L'opérateur **`%>`** trie dans l'ordre numérique décroissant, la liste donnée n argument.C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
`%>LISTE`
Le résultat est une liste des éléments de **`LISTE`** triés dans l'ordre numérique décroissant.
Mnémotechnique : Chaque élément est > au suivant

Exemple
`LISTE = %>{5,4,0,1,9,4}`
Le résultat est {9,5,4,4,1,0}

Opérateur %< (tri)

Opérateur %<

L'opérateur **`%<`** trie dans l'ordre numérique croissant, la liste donnée n argument.C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
`%<LISTE`
Le résultat est une liste des éléments de **`LISTE`** triés dans l'ordre numérique croissant.
Mnémotechnique : Chaque élément est < au suivant

Exemple
`LISTE = %<{5,4,0,1,9,4}`
Le résultat est {0,1,4,4,5,9}

Opérateur $> (tri)

Opérateur $>

L'opérateur $> trie dans l'ordre alphabétique décroissant, la liste donnée n argument.C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
$>LISTE
Le résultat est une liste des éléments de **LISTE** triés dans l'ordre alphabétique décroissant.
Mnémotechnique : Chaque élément est > au suivant

Exemple
```
LISTE = $>{"k",pm","az",ze"}
```
Le résultat est {"ze","pm",k","az"}

Opérateur $< (tri)

Opérateur $<

L'opérateur $< trie dans l'ordre alphabétique croissant, la liste donnée n argument.C'est un opérateur unaire à droite, c'est à dire qu'il ne possède qu'un seul argument placé à sa droite.

Syntaxe
$<LISTE
Le résultat est une liste des éléments de **LISTE** triés dans l'ordre alphabétique croissant.
Mnémotechnique : Chaque élément est < au suivant

Exemple
```
LISTE = $<{"k",pm","az",ze"}
```
Le résultat est {"az","k","pm","ze"}

Opérateur >> (décale à droite)

Opérateur >>

Cet opérateur décale d'un ou plusieurs rangs (argument de droite) vers la droite, les éléments d'une liste donnée en argument. Les éléments décalés sont purement et simplement éliminés, à la différence de l'opérateur rotation qui les fait cycler.

Un décalage d'un nombre supérieur ou égal au nombre d'éléments de la liste produit un liste vide. Un décalage négatif est sans effet.

Syntaxe
LISTE >> n
Le résultat est une liste des éléments de **LISTE** décalés de **n** rangs vers la droite.

Exemple
```
LISTE = {"k",pm","az",ze"}
item = LISTE >> 2
```
Le résultat est {"k","pm"}, les 2 éléments les plus à droite sont éjectés.

Opérateur << (décale à gauche)

Opérateur <<

Cet opérateur décale d'un ou plusieurs rangs (argument de droite) vers la gauche, les éléments d'une liste donnée en argument. Les éléments décalés sont purement et simplement éliminés, à la différence de l'opérateur rotation qui les fait cycler.

Un décalage d'un nombre supérieur ou égal au nombre d'éléments de la liste produit un liste vide. Un décalage négatif est sans effet.

Syntaxe
`LISTE << n`
Le résultat est une liste des éléments de **`LISTE`** décalés de **`n`** rangs vers la gauche.

Exemple
```
LISTE = {"k",pm","az",ze"}
item = LISTE << 1
```
Le résultat est {`pm","az",ze"`}, le 1er élément le plus à gauche est éjecté.

Opérateur <> (rotation)

Opérateur <>

Fait tourner les éléments d'une liste (opérande de gauche) par décalage à droite, le nombre de fois indiqué par l'opérande de droite. Les éléments qui "sortent" à droite rentrent par la gauche. On tourne dans l'autre sens avec des valeurs négatives. Le nombre d'éléments de la liste reste inchangé.

Syntaxe
`LISTE <> n`
Le résultat est la liste des éléments de **`LISTE`** dont les **`n`** éléments les plus à droite ont été déplacés devant les premiers éléments les plus à gauche.

Exemple
```
LISTE = {"k",pm","az",ze","tr"}
item = LISTE <> 2
```
Le résultat est {`ze","tr","k",pm","az"`}.

Opérateur == (égal) listes

Opérateur == (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si les deux listes sont identiques. Deux listes sont égales si et seulement chaque élément de même rang est identique, et les deux listes ont le même nombre d'éléments.

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 == liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur == dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} == {2,1}  est FAUX
{"ABC",{1,2}} == {"ABC",{1,2}} est VRAI
{1,{1,2}} == {{1,1},2} est FAUX
{1} == 1 est rejeté en erreur
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur != (différent) listes

Opérateur != (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si les deux listes sont différentes. Deux listes sont différentes si et seulement si au moins un élément de même rang est différent, ou si leur nombre d'éléments est différent.

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 == liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur != dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} != {2,1}  est VRAI
{"ABC",{1,2}} != {"ABC",{1,2}} est FAUX
{1,{1,2}} != {{1,1},2} est VRAI
{1} != 1 est rejeté en erreur
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur > (supérieur) listes

Opérateur > (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si la liste à gauche est strictement "supérieure" à la liste à droite.

Une liste A est "supérieure" à une liste B si et seulement si, en comparant les éléments de A et B au même rang :
- la première différence est un élément de A strictement supérieur à son correspondant dans B,
- ou la première différence est un élément A en plus par rapport à la liste B (liste A plus grande).

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 > liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur > dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} > {3,0}  est FAUX car 1 n'est pas > 3
{"A","ZAC"} > {"A","ABC"} est VRAI car la 1ère différence est un élément "ZAC"
> "ABC"
{1,2,3} > {1,2} est VRAI car tous les éléments comparables sont égaux, mais
liste de gauche plus longue
{1} > 1 est rejeté en erreur
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur >= (supérieur ou égal) listes

Opérateur >= (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si la liste à gauche est "supérieure" ou égale à la liste à droite.

Une liste A est "supérieure" ou égale à une liste B si et seulement si, en comparant les éléments de A et B au même rang :
- les deux listes sont égales
- ou la première différence est un élément de A strictement supérieur à son correspondant dans B,
- ou la première différence est un élément A en plus par rapport à la liste B (liste A plus grande).

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 >= liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur >= dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} >= {3,0}  est FAUX car 1 n'est pas > 3
{"A","ZAC"} >= {"A","ABC"} est VRAI car la 1ère différence est un élément
"ZAC" > "ABC"
{1,2,3} >= {1,2} est VRAI car tous les éléments comparables sont égaux, mais
liste de gauche plus longue
{1} >= 1 est rejeté en erreur
```

Voir aussi
opérateur <
opérateur <=
opérateur ==
opérateur !=
opérateur >=
opérateur >

Opérateur < (inférieur) listes

Opérateur < (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si la liste à gauche est strictement "inférieure" à la liste à droite.

Une liste A est "inférieure" à une liste B si et seulement si, en comparant les éléments de A et B au même rang :
- la première différence est un élément de A strictement inférieur à son correspondant dans B,
- ou la première différence est un élément A en moins par rapport à la liste B (liste A plus petite).

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 < liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur < dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} < {3,0}  est VRAI car 1 est < 3
{"A","ZAC"} < {"A","ABC"} est FAUX car la 1ère différence est un élément "ZAC"
> "ABC"
{1,2,3} < {1,2} est FAUX car tous les éléments comparables sont égaux, mais
liste de gauche plus longue
{1} > 1 est rejeté en erreur
```

Voir aussi
opérateur <

[opérateur <=]
[opérateur ==]
[opérateur !=]
[opérateur >=]
[opérateur >]

Opérateur <= (inférieur ou égal) listes

Opérateur <= (listes)

Cet opérateur compare deux listes et donne le résultat booléen VRAI si la liste à gauche est "inférieure" ou égale à la liste à droite.

Une liste A est "inférieure" ou égale à une liste B si et seulement si, en comparant les éléments de A et B au même rang :
- les deux listes sont égales
- ou la première différence est un élément de A strictement inférieur à son correspondant dans B,
- ou la première différence est un élément A en moins par rapport à la liste B (liste A plus petite).

Rappelons que le booléen VRAI est le nombre entier 1, et le booléen FAUX le nombre 0.

L'opérateur rejette en erreur une tentative de comparaison entre une liste et un élément simple

Syntaxe
liste1 <= liste2

Les éléments des deux listes sont comparés selon leur type (cf la description de l'opérateur <= dans le cas des arguments entiers, décimaux ou chaîne de caractères). L'opérateur accepte les listes imbriquées

Exemples
```
{1,2} <= {3,0}  est VRAI car 1 n'est pas > 3
{"A","ZAC"} <= {"A","ABC"} est FAUX car la 1ère différence est un élément
"ZAC" > "ABC"
{1,2,3} <= {1,2} est FAUX car tous les éléments comparables sont égaux, mais
liste de gauche plus longue
{1} >= 1 est rejeté en erreur
```

Voir aussi
[opérateur <]
[opérateur <=]
[opérateur ==]
[opérateur !=]
[opérateur >=]
[opérateur >]

°Distribution

La notation °distribution

On utilise la notation ° (rond), utilisée accolée immédiatement **devant** le nom d'un opérateur ou d'une fonction, pour signifier que cette opération ou fonction doit être distribuée sur chacun des éléments d'une ou plusieurs listes. Le résultat obtenu est toujours **une liste**. Cette °notation s'applique à :

- - n'importe quel opérateur binaire (+, -, *, &, >=, etc..),
- - n'importe quel opérateur unaire (-, !, $, etc..), que ce soit unaire droite ou unaire gauche,
- - les fonctions d'un, deux ou trois arguments, même les fonctions définies par l'utilisateur, à condition qu'elles renvoient une ou zéro valeur en retour.

Le principe d'une distribution est de faire agir sur des listes, une fonction ou un opérateur initialement conçu pour opérer sur des éléments simples.

De plus, pour les opérateurs ou fonctions binaires, les deux arguments peuvent être soit tous deux des listes, soit l'un une liste et l'autre un élément simple (cf les exemples mixtes).

Exemples (arguments listes)

`{"A","B"}  °+ {"C","D"} --> {"AC","BD"}`

`{1,2,3}  °+ {2,4,6} --> {3, 6, 9}`

`{2,3,4} °! --> {2,6,24}` (distribution de la fonction "factorielle" qui est unaire à gauche)

`°-{"azer", 8, 1.23 } --> { "reza",-8, -1.23 }` (distribution de la fonction "opposé" qui est unaire à droite)

`°Lettre({1,2,3}) --> {"A", "B", "C"}`

`°Inclus({"AF","DEF",DG"},"DEFGH") --> {0,1,1}`

Exemples mixtes (un argument liste, un argument simple)

`{"A","B"} °+ "X" --> {"AX","BX"}`

`10 °+ {1,2,3} --> {11, 12, 13}`

Enfin, il est possible de bloquer la distribution sur un argument liste.

Voir Opérateur @

Opérateur @ (anti-distribution)

L'opérateur @ (anti-distribution)

L'opérateur ° (rond) permet de distribuer un opérateur ou une fonction sur chaque élément d'une liste. Quand l'opérateur ou la fonction possède plusieurs arguments, la distribution opère sur **tous les arguments de type liste.**
Or Il est parfois utile d'empêcher la distribution sur un argument de type liste. C'est le rôle de l'opérateur unaire @

Par exemple :

```
FELINS = {"lion","panthère","tigre"}
LISTEANIMAUX = { {"escargot","lion","tigre"}, {"limace"},{"dauphin","lion"}}
```

On désire obtenir `LISTEFELINS = { {"lion","tigre"}, {}, {"lion"} }`

en ne conservant dans les 3 sous-listes de LISTEANIMAUX que ceux appartenant à FELINS.

Une distribution de l'intersection Inter() semble s'imposer.

Or :

`LISTEFELINS = °Inter(LISTEANIMAUX,FELINS)`
donne une erreur "arguments de la fonction Inter invalides" car elle conduit à évaluer, pour commencer : `Inter( {"escargot",lion","tigre"} , "lion" )` ce qui est illégal, puisque "lion" n'est pas une liste, mais le 1er élément de la liste FELINS qui est distribuée. Il faut empêcher la distribution sur la liste FELINS. Pour celà, on va utiliser l'opérateur @ :

`LISTEFELINS = °Inter(LISTEANIMAUX,@FELINS)`
qui donnera bien le résultat prévu.

Réduction°

La notation réduction°

On utilise la notation ° (rond), utilisée accolée immédiatement **après** le nom d'un opérateur ou d'une fonction, pour signifier que cette opération ou fonction **binaire** doit être opérée sur chacun des éléments de la liste donnée en unique argument. La fonction commence à agir avec les éléments 1 et 2. Puis ce résultat intermédiaire agit avec l'élément 3. Puis ce résultat intermédiaire agit avec l'élément 4. Ainsi de suite jusqu'au dernier élément. Le résultat obtenu est toujours un **élément** simple.

Si la liste comporte un seul élément, le résultat est cet élément. Si la liste est vide, le résultat est une liste vide.

Cette notation° s'applique à :

- - n'importe quel opérateur binaire (+, -, *, &&, >=, etc..) :
- - les fonctions de deux arguments, même les fonctions définies par l'utilisateur, à condition qu'elles ne renvoient qu'une valeur en retour.

Le principe de la réduction est d'étendre à un nombre quelconque d'arguments une fonction ou un opérateur initialement conçu pour 2 arguments : +° calcule la somme des arguments d'une liste, &&° réalise le "ET logique" entre les éléments d'une liste, etc...

Exemples

```
+°{"C","DE"} --> "CDE"

&&°{1,0,1,1} --> 0

+°{2,4,6} --> 12

%Fonction(x,y)
%Return(10*x+y)

Fonction°({1,4,2,8,5,7}) --> 142857
```

°Produit externe°

La notation °produit externe°

On utilise la notation ° (rond), utilisée accolée immédiatement **avant** et **après** le nom d'un opérateur ou d'une fonction, pour signifier que cette opération ou fonction doit être distribuée sur chacun des **couples** possibles d'éléments des deux listes en argument. Le résultat obtenu est toujours **une liste** dont le nombre d'éléments est le **produit** des nombres d'éléments des deux listes. Cette °notation° s'applique à :

- - n'importe quel opérateur binaire (+, -, *, &, >=, etc..) :
- - les fonctions de deux arguments, même les fonctions définies par l'utilisateur, à condition qu'elles renvoient une ou zéro valeur en retour.

Les deux arguments peuvent être soit tous deux des listes, soit l'un une liste et l'autre un élément simple.

Le but du produit externe est de calculer un résultat selon deux dimensions (en lignes et colonnes, respectivement représentées par les arguments droite et gauches de l'opérateur), un peu à la manière d'une formule Excel appliquée sur un tableau.

Exemples

```
{"A","B"}  °+° {"C","D"} --> {"AC","AD","BC","BD"}

{1,2,3}  °*° {5,6,7,8} --> {5,6,7,8,10,12,14,16,15,18,21,24}

°Debut°({"ABC","DEF"},{1,2,3}) --> {"A","AB","ABC","D","DE","DEF"}

°Debut°({"ABC","DEF","GH"},2) --> {"AB","DE","GH"}
```

une matrice identité n x n est très simplement construite avec un produit externe : (%n °==° %n)µn

Diverses

Fonctions diverses

Fonctions de changement de type de variable

Chaine
Entier
Flottant
Hexa

Autres

Trialpha	Tri des éléments d'un tableau dans l'ordre alphabétique
Trinum	Tri des éléments d'un tableau dans l'ordre numérique
Exit	Fin de fonction ou fin de programme
Evalue	Evalue une expression
Heure	Donne l'heure
Wait	Attend une ou plusieurs valeurs futures

Fonctions de communication

Breakpoint	Point d'arrêt conditionnel
Ecrirefichier	Ecrit une ligne dans un fichier

Fermefichier	Libère un fichier pour réemploi dans le même script
Lirefichier	instruction-bloc de lecture itérative d'un fichier
Message	Ecrit une ligne en zone message
Messagebox	Ecrit une ligne dans une boîte de message windows
Pause	Suspend momentanément un programme EEA
Saisie	Affiche une boîte windows de saisie d'un texte
Saisiefichier	Affiche une boîte wondows de choix d'un fichier
Selection	Ecrit une ligne dans la zone liste de mots
Trace	Modifie le flag trace
Tracemessage	Affiche un texte dans un champ fixe

Breakpoint

Breakpoint(chaine,condition)

L'instruction **Breakpoint** permet d'interrompre conditionnellement le programme EEA, selon qu'une condition est vraie ou fausse.

Le 1er argument est une simple chaîne de caractère, le second la condition à évaluer à VRAI (= l'entier 1) pour que le programme soit interrompu et affiche le premier argument en zone message. C'est une instruction très utile pour la mise au point de vos programme EEA récalcitrants.

Syntaxe
Breakpoint(texte,condition)
Il n'y a pas de valeur retour.

Exemple :

```
for (x=1; x<10 ;x += 1)
Breakpoint("arrêt du programme",x == 5)
endfor
```

le programme se déroule et incrémente la variable x de 1 à 10. Cependant, à la valeur 5, l'instruction provoque l'interruption du programme, et l'ajout de la ligne :

[Breakpoint] `arrêt du programme`

dans la fenêtre de messages. Il est alors possible de poursuivre l'exécution du programme en appuyant sur le bouton vert "Poursuivre" (le bouton en haut à droite), ce qui provoque la poursuite de la boucle jusqu'à la fin de boucle à x = 10.

Chaine

Chaine(expression)

Cette fonction permet de convertir un nombre entier ou un nombre décimal en une chaîne de caractères correspondant à sa représentation numérique en base 10.

Syntaxe
Chaine(expression)

Exemples

a = Chaine(345)
a est la chaîne de caractères "345"

a = Chaine(12.34 + 3.21)
a est égal à la chaîne "15.55"

Voir aussi
Format

Entier

Entier(expression)

La fonction **Entier** permet de convertir soit une chaîne de caractères ne contenant que des chiffres en la valeur numérique correspondante, soit un nombre décimal en sa partie entière. Si l'argument de cette fonction ne contient pas que des chiffres, le résultat est l'entier 0.

Syntaxe
Entier(expression)

Exemples
a = Entier("345")
le résultat est le nombre entier 345.

a = "12"
b = "34"
c = Entier(a+b)
d = Entier(a)+Entier(b)
le contenu de la variable c est le nombre 1234, car "12" + "34" est la concaténation de deux chaînes de caractères, avec pour valeur "1234" dont la conversion est ne nombre 1234.
le contenu de la variable d est le nombre 46, addition arithmétique de 12 et 34, conversions respectives de "12" et "34".

a = Entier(3.14159)
le résultat est le nombre entier 3, qui est la partie entière de 3.14159

a = Entier("Deux")
le résultat est 0

Voir aussi
Chaine
Flottant

Evalue

Evalue(expression)

La fonction **Evalue** permet d'interpréter et d'exécuter une instruction (une seule) passée en argument. L'argument est une chaîne de caractères. Il n'y a pas de valeur retour. Attention l'expression utilisée doit correspondre en tout points à une instruction AJL valide. Les instructions de bloc ne sont pas admises. les fonctions et toutes les variables peuvent être utilisées dans l'expression.

Syntaxe
Evalue(expression)

Exemples incorrects
//un résultat de calcul n'est pas une instruction correcte
Evalue("1+1")

//une instruction de bloc ne peut être utilisée, même si la fin de bloc est spécifiée plus loin.
Evalue ("si(x == 0)")
finsi

Exemples corrects
//expressions simples

Evalue ("x = 1 + 1")

Evalue("x = 2^5*9^2")
Selection(x)

L'expression "x = 2^5*9^2" est évalue, et en conséquence une instruction "x = 2592" est exécutée. La valeur de x est persistante exactement comme si l'instruction
x = 2^5*9^2
avait été directement exécutée

//expression compliquée
%f(x,y)
%return(5*x-2*y)

x = 2
y= 4
Evalue("z = f(x,y)")

comme prévu, tout se passe comme si z = f(2,4) = 2 avait été exécutée. La valeur de z est persistante et peut être réemployée.

Existe

Existe(variable[,défaut])

La fonction Existe permet de déterminer si une variable ou un élément de tableau a déjà été créé par l'exécution d'une instruction d'affectation.
La forme à 2 arguments permet d'obtenir soit la valeur de la variable si elle existe, soit une valeur par défaut dans le cas contraire.

Syntaxes
`Existe(nom)`
`nom` est un nom de variable ou un élément de tableau. Le résultat de cette fonction est un VRAI ou FAUX (c'est à dire les entiers 1 ou 0) selon que l'élément cherché existe ou non.

`Existe(nom,defaut)`

`nom` est un nom de variable ou un élément de tableau. Le résultat de cette fonction est `defaut` si l'élément cherché n'existe pas, ou la valeur stockée dans `nom` si `nom` existe.

Exemple

```
table[indice] = Existe(table[indice],0) + 1
```

Dans cet exemple, le programme teste l'existence de l'élément de tableau `indice` dans le tableau `table`. Si l'élément existe, il lui ajoute la valeur 1, sinon il le crée en lui affectant la valeur 0 + 1 = 1.

Note sur la fonction Existe
Cette fonction, *sans doute la plus puissante proposée par EEA*, remplace à elle seule une base de donnée. Par exemple, imaginez que vous ayez besoin de constituer une tableau de n mots TMots, de TMots[1] à TMots[n]. Vous valorisez les éléments de tableau comme ceci :

```
TMots[1] ="TOTO"
TMots[2] ="TATA"
etc...
```

La méthode traditionnelle pour savoir si un mot donné, stocké dans une variable MOT, est ou non dans le tableau est de coder une boucle de recherche :

```
boucle(i=1; i<=n ;i+=1)
  si (TMots[i] == MOT)
    //le mot est trouvé à l'indice i
  finsi
finboucle
```

Avec la fonction Existe, il suffit de mettre les mots stockés en INDICE du tableau et non pas en valeur..

```
TMots["TOTO"] = 1
TMots["TATA"] = 2
etc...
```

Ensuite, vous utiliserez le tableau comme une base de données : une instruction suffit pour retrouver (ou non) le mot cherché dans la table :

```
si(Existe(TMots[MOT]))
    //le mot est trouvé. la valeur stockée dans le tableau TMots[MOT] est
immédiatement utilisable.
finsi
```

voir Dictionnaires python

Flottant

Flottant(expression)

La fonction Flottant permet de convertir une chaîne de caractères ne contenant que des chiffres et éventuellement un point en la valeur numérique en virgule flottante correspondante. Elle peut aussi s'appliquer à un nombre entier, qu'elle convertit de même en sa valeur sous forme virgule flottante.

Syntaxe
Flottant(expression)

Exemple
a = Flottant("3.45")
affecte à la variable a le nombre flottant 3.45.

a = Flottant(3)
affecte à la variable a le nombre flottant 3.0.

Si l'argument de cette fonction ne contient pas que des chiffres, le résultat est 0.0

Voir aussi
Chaine
Entier

Heure

Heure()

La fonction **Heure** renvoie l'heure courante, selon le format hh:mm:ss.
Il n'y a pas d'argument d'entrée, et la valeur retour est une chaîne de caractères.

Syntaxe
Heure()

Exemple
```
Message("Il est "+Heure())
```


Hexa

Hexa(expression)

La fonction **Hexa** permet de convertir en nombre entier soit une chaîne de caractères ne contenant que des chiffres, soit un nombre décimal, soit un nombre entier écrit en base 10. Si l'argument de cette fonction ne contient pas que des chiffres, le résultat est l'entier 0. De plus l'affichage ultérieur par défaut de ce nombre se fera selon la conversion hexadécimale.

Syntaxe
Hexa(expression)

Exemples
```
a = Hexa("128")
```
le résultat est le nombre entier 128, dont la représentation, hexadécimale est #80.

de même :
```
a = Hexa(128)
```
le résultat est le nombre entier 128, dont la représentation, hexadécimale est #80

```
Message(a+10)
```
affichera :
```
#8A
```

Voir aussi
Chaine

Flottant

Ecrirefichier

Ecrirefichier(fichier, ligne)

La fonction Ecrirefichier permet l'écriture ligne à ligne d'un fichier texte. Le résultat du premier argument est interprété comme chaîne de caractères pour obtenir le nom de fichier. Le second argument est interprété comme la valeur de la ligne de caractères à écrire dans ce fichier.

La première exécution de cette fonction provoque l'ouverture en écriture du fichier dont le nom est donné par la valeur du premier argument. Il sera donc réservé par AJL jusqu'à la fin de l'exécution du programme EEA.

- Si le fichier n'existe pas, il est créé avec le nom spécifié
- Si le fichier existe déjà, un message de confirmation est envoyé avant toute modification du fichier. Si vous confirmez, le fichier est alors effacé, et sera irrémédiablement remplacé par le contenu spécifié par votre programme EEA; Si vous ne confirmez pas, le programme EEA s'arrête immédiatement.
- Si le programme EEA est autorisé à continuer, une première ligne de texte est ajoutée au fichier vide.
- Toutes les occurrences suivantes de toute instruction Ecrirefichier portant sur le même nom de fichier, où qu'elles se trouvent dans votre programme, provoqueront l'ajout de nouvelles lignes à la suite les unes des autres. Le fichier est automatiquement refermé et libéré à la fin du programme EEA. Si le programme est momentanément interrompu, le fichier n'est pas libéré et EEA autorisera l'ajout de nouvelles lignes dès que le programme EEA poursuivra son exécution.

Syntaxe

```
Ecrirefichier(fichier,enregistrement)
```

les deux arguments sont de type chaîne. Il n'y a pas de valeur rendue en retour.

Note

Ecrirefichier est une simple fonction EEA. A ce titre, et comme pour toutes les fonctions, le séparateur d'arguments est une virgule. On remarquera que Lirefichier est une instruction de bloc (elle fonctionne associée à une instruction finlirefichier). A ce titre, et comme toutes les instructions de bloc, le séparateur d'arguments est un point-virgule.

Exemple

```
NomDeFichier = Saisiefichier("Quel fichier faut-il créer?")
for (i = 1; i <= 10 ; i += 1)
  Ecrirefichier(NomDeFichier, ligne[i])
endfor
```

Voir aussi

Lirefichier
Saisiefichier
Fermefichier

Fermefichier

Fermefichier(fichier)

La fonction **Fermefichier** sert à libérer les verrous posés par AJL lors des fonctions de lecture ou écriture que sont **lirefichier** et **Ecrirefichier**.

Vous n'avez normalement pas besoin d'utiliser **Fermefichier** si vous employez un fichier seulement dans

un seul bloc **lirefichier**, ou dans une seule boucle de fonction **Ecrirefichier**. Cependant, il est parfois utile de créer un fichier avec des **Ecrirefichier**, puis de le relire avec des fonctions **lirefichier** ou même **Chargerdico**. Dans un tel cas, il est nécessaire d'utiliser un **Fermefichier**

Syntaxe

```
Fermefichier(fichier)
```

L'argument est de type chaîne et désigne le nom du fichier dont il faut libérer les verrous. Il n'y a pas de valeur rendue en retour.

Exemple d'utilisation

```
//phase 1 : créer un fichier dictionnaire avec une liste de
mots
Nom = Saisiefichier("Quel dictionnaire faut-il créer?")
Dico = {"BIDULE","MACHIN","TRUC"}
°Ecrirefichier(NomDeFichier,Dico)

//phase 2 : indiquer à AJL que la série d'écritures est
terminée
Fermefichier(Nom)

//phase 3 : recharger le fichier en tant que dictionnaire
Chargerdico(Nom)
```

Voir aussi
Lirefichier
Saisiefichier
Ecrirefichier
Chargerdico

Message

Message(chaine)

L'instruction **Message** permet d'afficher un message dans la zone message, cadre du bas dans fenêtre principale de EEA.
L'argument est considéré comme une chaîne de caractère interprétée avec les mêmes possibilités de formatage automatique que celles de la fonction Format
La longueur maximale d'un texte affiché par cette fonction est fixée à **100 caractères**. La zone message quant à elle est limitée à 1000 lignes, les plus récentes remplaçant les plus anciennes si le nombre maximal de 1000 lignes est atteint.

Synonyme
Display

Syntaxe
Message(chaine)
Il n'y a pas de valeur retour.

Exemples

```
Message("coucou!")
```

résultat : le texte "coucou!" est affiché dans la zone message

```
a = "cou"
```

```
b = "!"
Message(2*a+b)
```

résultat : le texte "coucou!" est affiché dans la zone message

Exemple utilisant la possibilité de formatage automatique

```
ccc = "jlkhg"
ddd = 3.1415
eee = 1

Message("La chaîne = %s, l'entier = %i, le décimal = %f",ccc,eee,ddd)
```

résultat la ligne ci-dessous est affichée :

```
La chaîne = jlkhg, l'entier = 1, le décimal = 3.141500
```


Messagebox

Messagebox(chaine)

L'instruction **Messagebox** permet d'afficher un message dans une boîte de dialogue classique Windows. Il est nécessaire d'appuyer sur le bouton OK de celle boîte de dialogue pour que le programme EEA se poursuive. L'argument est considéré comme une chaîne de caractère.

Syntaxe
Messagebox(chaine)
Il n'y a pas de valeur retour.

Exemples
```
Messagebox("coucou!")
```

résultat : le texte "coucou!" est affiché dans une boîte de dialogue de style information Windows, de type modale.

Pause

Pause(chaine)

L'instruction **Pause** permet d'interrompre momentanément le programme EEA, et d'ajouter une ligne de texte dans la fenêtre de message. Elle est strictement équivalente à l'appui manuel sur le **bouton Interrompre**, qui est actif pendant le déroulement d'un script EEA.

L'argument est considéré comme une chaîne de caractère interprétée avec les mêmes possibilités de formatage automatique que celles de la fonction Format

Syntaxe
Pause(texte)
Il n'y a pas de valeur retour.

Exemple :
mot = "tintinnabuler"
Pause("J'ai trouvé ce mot très remarquable "+mot)

provoque l'interruption du programme, et l'ajout de la ligne :

[Pause] J'ai trouvé ce mot très remarquable tintinnabuler

dans la fenêtre de messages.

Il est alors possible de poursuivre l'exécution du programme en appuyant sur le bouton vert "Poursuivre" (le bouton en haut à droite).

Exemple utilisant la possibilité de formatage automatique

```
ccc = "jlkhg"
ddd = 3.1415
eee = 1

Pause("La chaîne = %s, l'entier = %i, le décimal = %f",ccc,eee,ddd)
```

résultat la ligne ci-dessous est affichée :

```
[Pause] La chaîne = jlkhg, l'entier = 1, le décimal = 3.141500
```


Saisie

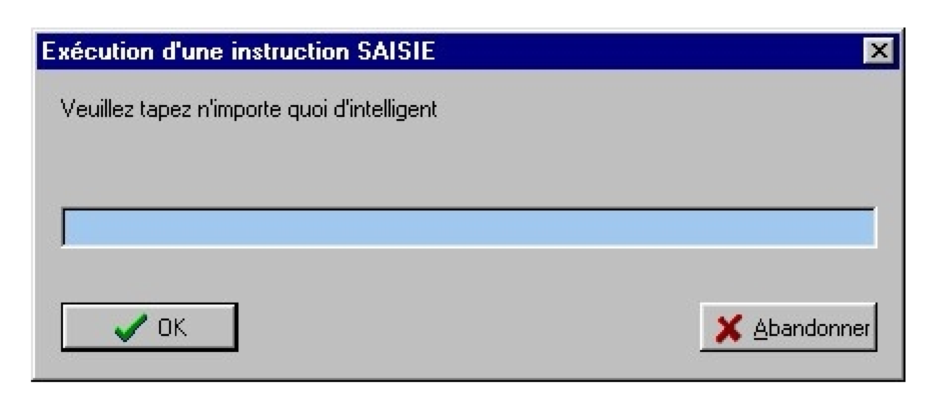
Saisie(texte)

La fonction **Saisie** affiche une boîte de dialogue Windows, ornée du texte composé de la chaîne de caractère passée en argument. De plus, cette boite de dialogue comporte une zone de saisie (quelle surprise !) permettant à l'utilisateur, ainsi invité à s'exprimer, d'y démontrer ses talents de dactylographie. Le résultat de la fonction est tout simplement le texte tapé dans cette zone. Si vous appuyez sur la touche Abandonner au lieu de OK, le résultat de la saisie est la chaîne vide "".

Syntaxe
Saisie(expression)

Exemples
a = Saisie("Veuillez taper n'importe quoi d'intelligent")

provoque dans un premier temps l'apparition de :



Ce qui est assez traumatisant je l'avoue. 1 - 0 pour l'ordinateur.
Mettons que vous tapiez **"Ah, je ris de me voir si belle en ce miroir"** dans la zone de saisie. Très belle réplique, 1 partout, non mais sans blagues.

Dès que vous appuierez sur le bouton OK, ce texte impérissable sera le résultat de la fonction, et donc transféré dans la variable a. Notez que si c'est un chiffre que vous demandez, il vous suffit de faire :

a = Entier(Saisie("Veuillez taper le code de votre carte bleue"))

nb : çà marche aussi avec une carte American Express.

Saisiefichier

Saisiefichier(texte)

La fonction **Saisiefichier** affiche une boîte de dialogue standard de choix de fichier, dont le titre est composé de la chaîne de caractère passé en argument. Si vous cliquez sur Abandonner au lieu de OK, le résultat de la fonction est la chaîne vide "". Il vous appartient de traiter cette éventualité dans vos scripts. cf l'exemple.

Syntaxe
Saisiefichier(expression)
Si vous utilisez cette boîte pour choisir un fichier et cliquez sur OK, le résultat de la fonction est le nom de fichier choisi.

Exemple
```
//provoque l'apparition de la boite de choix de fichier.
a = Saisiefichier("Choix du fichier à explorer : ")
//si l'utilisateur a abandonné la saisie, alors fin de script
si (a == "")
  exit
finsi
```

Voir aussi
Bloc lirefichier.

Selection

Selection(texte)

L'instruction Selection permet d'ajouter un texte à la liste des mots trouvés, dans la fenêtre principale de AJL.

Optionnellement, l'argument est interprété avec les mêmes possibilités de formatage automatique que celles de la fonction Format

Syntaxe
Selection(chaine)

Exemple
Selection("Eve le samedi se repose")

Exemple utilisant la possibilité de formatage automatique

```
ccc = "jlkhg"
ddd = 3.1415
eee = 1

Selection("La chaîne = %s, l'entier = %i, le décimal = %f",ccc,eee,ddd)
```

résultat: la ligne ci-dessous est ajoutée à la liste affichée :

```
La chaîne = jlkhg, l'entier = 1, le décimal = 3.141500
```


Trialpha

Trialpha(nom,premier,dernier)

La fonction **Trialpha** trie dans l'ordre alphabétique les éléments de tableau dont le nom est précisé par le premier argument. Les éléments triés sont donnés par les indices numériques du premier et du dernier élément de tableau, spécifiés respectivement en second et dernier arguments

Syntaxe
Trialpha(nom,premier,dernier)
Le premier argument est un nom de tableau, les deux autres des nombres entiers. il n'y a pas de valeur retour.

Exemples

```
Tableau[1] = "A"
Tableau[2] = "Z"
Tableau[3] = "E"
Tableau[4] = "R"
Tableau[5] = "T"
Tableau[6] = "Y"
Trialpha(Tableau,2,4)
```

seuls les éléments de tableau d'indice 2,3,4 sont triés, et donc désormais :

```
Tableau[1] = "A"
Tableau[2] = "E"
Tableau[3] = "R"
Tableau[4] = "Z"
Tableau[5] = "T"
Tableau[6] = "Y"
```

autre exemple:

```
a = Saisie("Veuillez taper une suite quelconque de lettres à trier")
x = Dissocie(Tab,a)
Trialpha(Tab,1,x)
a = Associe(Tab,x)
```

Ce petit script demande la saisie d'une suite de lettres, les associe à des éléments de tableau, les trie, puis les ré-associe dans une variable finale.

Voir aussi
Dissocie

Trace

Trace(bool)

L'instruction **Trace** permet de choisir la valeur du flag "Trace", qui est positionné dans l'interface AJL par une case à cocher "Tracer" situer au dessus de la zone Selection.

L' argument est une valeur booléenne 1 pour activer la trace, 0 pour la désactiver.

Syntaxe
Trace(valeur)

Il n'y a pas de valeur retour.

Exemple :

```
for (x=1; x<10 ;x += 1)
  Trace(x==5)
endfor
```

Le programme se déroule et incrémente la variable x de 1 à 10. Cependant, à la valeur 5, l'instruction provoque le démarrage de la trace automatique, qui affichera donc

Compilation terminée, exécution en cours...
[Trace] x = 6
Exécution terminée. Durée: 0.001 secondes

En effet, à x = 5, la trace est activée. Elle est active pour l'affectation suivante de la boucle for, qui donne x = 6. A x=6, la trace est désactivée par l'exécution de Trace(6 == 5) qui donne une valeur Trace(0).

Tracemessage

Tracemessage(chaine)

L'instruction **Tracemessage** permet d'afficher un message dans le champ où en mode édition, on trouve le numéro de la ligne courante, entre les deux cadres de la fenêtre principale de EEA.
L'intérêt de cette instruction est double :
- elle ne "consomme" pas une ligne de texte, à la différence des instructions `Selection` ou `Message`, car c'est toujours le même champ qui est employé pour contenir le texte à afficher.
- elle est beaucoup plus rapide que les instructions `Message` ou `Selection`.

L'argument est considéré comme une chaîne de caractère interprétée avec les mêmes possibilités de formatage automatique que celles de la fonction Format

Syntaxe
Tracemessage(chaine)
Il n'y a pas de valeur retour.

Exemples
```
Tracemessage("coucou!")
```

résultat : le texte "coucou!" est affiché.

Trinum

Trinum(nom,premier,dernier)

La fonction **Trinum** trie numériquement les éléments de tableau dont le nom est précisé par le premier argument. Les éléments triés sont donnés par les indices numériques du premier et du dernier élément de tableau, spécifiés respectivement en second et dernier arguments

Syntaxe
Trinum(nom,premier,dernier)

Le premier argument est un nom de tableau, les deux autres des nombres entiers. il n'y a pas de valeur retour.

Exemples
```
Tableau[1] = 1
Tableau[2] = 4
Tableau[3] = 2
Tableau[4] = 8
Tableau[5] = 5
Tableau[6] = 7
Trinum(Tableau,2,5)
```
les éléments de tableau d'indice 2,3,4,5 sont triés, et donc désormais :
```
Tableau[1] = 1
Tableau[2] = 2
Tableau[3] = 4
Tableau[4] = 5
Tableau[5] = 8
Tableau[6] = 7
```


Exit

Exit

Cette instruction a deux fonctions, selon qu'elle est utilisée dans un programme EEA simple, ou à l'intérieur d'une fonction définie par vos soins.

- dans un programme EEA principal, l'exécution de cette instruction provoque l'arrêt immédiat du programme.
- à l'intérieur d'une fonction, cette instruction provoque l'arrêt de la fonction et l'exécution immédiate de sa clause de sortie %retour()

Syntaxe
Exit

Voir aussi
bloc de définition de fonction

Wait

Wait(...série d'arguments...)

Cette fonction permet d'attendre la bonne fin d'un ou plusieurs threads, en attendant la disponibilité des valeurs retours (dites valeurs futures) que ces fonctions thread renvoient. Cette fonction n'a aucune valeur retour.

Syntaxe
Wait(futur1, futur2, futur3)

Exemple

```
%faitquelquechose(message) thread
  Selection("on fait " + message)
%return(1)
```

```
v1 = faitquelquechose("ceci")
v2 = faitquelquechose("cela")

Wait(v1,v2)

Selection("nous avons terminé de faire ceci et cela")
```

Voir aussi
La programmation multi-tâches

Fonctions dictionnaire

Fonctions dictionnaire

Ces fonctions font appel ou sont liées au dictionnaire chargé dans AJL.

Anagramme	Calcule le nombre d'anagrammes réalisables avec un jeu de lettres
Anagrammeliste	Stocke dans une liste les anagrammes réalisables avec un jeu de lettres
Anagrammelistethread	Version multi-thread de la fonction précédente
Chargerdico	Change le dictionnaire en cours
Debutmot	Vérifie si une suite de lettres est un début d'un mot au dictionnaire
Dicoliste	Charger tout le dictionnaire dans une liste
Lexique	Vérifie si un mot est constructible avec un lexique donné
Liredico	instruction-bloc de lecture itérative d'un dictionnaire
Mot	Vérifie si une chaîne de caractères est un mot du dictionnaire
Tiragemot	Tire au sort un mot du dictionnaire

Anagramme

Anagramme(motsmin,motsmax,lgmin,lgmax,lettres)

La fonction **Anagramme** donne le nombre d'anagrammes possibles avec un jeu de lettre donné. Les deux premiers arguments numériques sont respectivement le nombre minimum et le nombre maximum de mots que devra comporter l'anagramme.Les deux suivants les longueurs minimales et maximales des mots employés dans l'anagramme. Le dernier argument est le jeu de lettres à employer. Le résultat est un nombre entier, correspondant au nombre de possibilités d'anagrammes découvertes.

Cette fonction permet avant tout de sélectionner les configurations de lettres intéressantes, voyez l'exemple ci-dessous.

NOTA : la fonction Anagramme ne fait pas appel au calcul parallèle (multi-threading), à la différence de l'interface utilisateur de l'onglet Anagramme. **L'utilisation de cette fonction est à réserver à la résolution d'anagrammes dont la calcul est rapide (largement moins de 1/10 seconde de calcul par anagramme).**

Syntaxe
Anagramme(min, max, lgmin, lgmax, lettres)

Le résultat est de type entier, les quatre premiers arguments sont entiers, et lettres est de type chaîne.

Exemples

nb = Anagramme(2,2,0,99,"MONACOBRESIL")

On recherche donc les anagrammes portant sur 2 mots exactement, sans contrainte de longueur pour les mots. Avec un dictionnaire réduit aux noms des pays, le résultat obtenu : nb = 2, est prometteur. En effet, on a deux anagrammes sur exactement deux mots : "MONACO" "BRESIL" et "COMORES" "LIBAN".

nb = Anagramme(2,2,0,99,"FRANCEITALIE")

Le résultat est seulement 1, Cette configuration est donc peu intéressante, puisqu'elle ne donne que le trivial FRANCE ITALIE.

Avertissement
non compatibilité de syntaxe avec les versions précédentes, antérieures à AJL 5.0, mais résultat identique.

Voir aussi
Anagrammeliste

Anagrammeliste

Anagrammeliste(nbmax,motsmin,motsmax,lgmin,lgmax,lettres)

La fonction **Anagrammeliste** renvoie une liste des anagrammes possibles d'un jeu de lettre donné.

Le premier argument est le nombre maximum d'anagrammes que la liste pourra stocker. Les 5 arguments suivants sont identiques à ceux de la fonction Anagramme, respectivement :
- nombre minimum et le nombre maximum de mots que devra comporter l'anagramme.
- longueurs minimales et maximales des mots employés dans l'anagramme,
- jeu de lettres à utiliser

La fonction renvoie la liste des anagrammes en valeur retour. Chaque anagramme est rendue sous forme d'une chaîne de caractères composée de la suite des mots trouvés, suivis chacun par un espace. Si le nombre d'anagrammes possibles excède le nombre précisé en 2ème argument, le nombre d'éléments de liste enregistrée est tronqué à la valeur maximum prévue par cet argument.

NOTA : la fonction Anagrammeliste ne fait pas appel au calcul parallèle (multi-threading), à la différence de l'interface utilisateur de l'onglet Anagramme. **L'utilisation de cette fonction est à réserver à la résolution d'anagrammes dont la calcul est rapide (largement moins de 1/10 seconde de calcul par anagramme).**

Sinon, on privilègira la fonction Anagrammelistethread.

Il est envisageable de gérer, avec cette fonction, des listes possédant plus d'un million de membres.

Syntaxe
```
Anagrammeliste(nb,min, max, lgmin, lgmax, lettres)
```

Exemples

```
L = Anagrammeliste(100,2,2,0,99,"MONACOBRESIL")
```

On recherche donc ici un maximum de 100 anagrammes portant sur 2 mots exactement, sans contrainte de longueur pour les mots. Avec un dictionnaire réduit aux noms des pays, la liste L prendra la valeur :

```
L = {"MONACO BRESIL ",  "COMORES LIBAN "}
```

pour rappel, l'expression
```
L%
```
renverra directement le nombre d'éléments de la liste. Exemple :

```
L = Anagrammeliste(1000000,4,4,6,10,"FORMIDABLEAIDEAUXJEUXDELETTRES")
Message(L%)
```

Ce script calcule les anagrammes demandées dans une liste L pouvant accueillir jusqu'à 1.000.000 membres. Le nombre de membres obtenus est affiché en zone message. Avec le dictionnaire scrabble standard, on en trouve 190256.

Voir aussi
Anagramme
Anagrammelistethread

Anagrammelistethread

Anagrammelistethread(nbmax,motsmin,motsmax,lgmin,lgmax,lettres)

La fonction **Anagrammelistethread** renvoie une liste des anagrammes possibles d'un jeu de lettre donné.

Le premier argument est le nombre maximum d'anagrammes que la liste pourra stocker. Les 5 arguments suivants sont identiques à ceux de la fonction Anagramme, respectivement :
- nombre minimum et le nombre maximum de mots que devra comporter l'anagramme.
- longueurs minimales et maximales des mots employés dans l'anagramme,
- jeu de lettres à utiliser

La fonction renvoie la liste des anagrammes en valeur retour. Chaque anagramme est rendue sous forme d'une chaîne de caractères composée de la suite des mots trouvés, suivis chacun par un espace. Si le nombre d'anagrammes possibles excède le nombre précisé en 2ème argument, le nombre d'éléments de liste enregistrée est tronqué à la valeur maximum prévue par cet argument.

NOTA : la fonction Anagrammelistethread fait largement appel au calcul parallèle (multi-threading), tout comme l'interface utilisateur de l'onglet Anagramme. **L'utilisation de cette fonction est à réserver à la résolution d'anagrammes complexes (une seconde ou plus de calcul)**, pour lesquels le gain de performance compense largement le coût de lancement et de gestion du calcul parallèle.

Sinon, on utilisera la version mono-thread Anagrammeliste.

Il est envisageable de gérer, avec cette fonction, des listes possédant plus d'un million de membres.

Syntaxe
```
Anagrammelistethread(nb,min, max, lgmin, lgmax, lettres)
```

Exemples

```
L = Anagrammelistethread(100,2,2,0,99,"MONACOBRESIL")
```

On recherche donc ici un maximum de 100 anagrammes portant sur 2 mots exactement, sans contrainte de longueur pour les mots. Avec un dictionnaire réduit aux noms des pays, la liste L prendra la valeur :

```
L = {"MONACO BRESIL ",  "COMORES LIBAN "}
```

pour rappel, l'expression
```
L%
```

renverra directement le nombre d'éléments de la liste. Exemple :

```
L = Anagrammelistethread(1000000,4,4,6,10,"FORMIDABLEAIDEAUXJEUXDELETTRES")
Message(L%)
```

Ce script calcule les anagrammes demandées dans une liste L pouvant accueillir jusqu'à 1.000.000 membres. Le nombre de membres obtenus est affiché en zone message. Avec le dictionnaire scrabble standard, on en trouve 190256.

Voir aussi
Anagramme
Anagrammeliste

Chargerdico

Chargerdico(dictionnaire)

Cette instruction permet de remplacer le ou les dictionnaires alphabétiques en cours par d'autres, exactement comme avec le dialogue accessible par le menu Fichier/Dictionnaires Alphabétiques. Les dictionnaires phonétiques ne sont pas pris en charge par cette fonction.

Le ou les dictionnaires ainsi chargés ne seront actifs que le temps de l'exécution du programme EEA. A la fin de celui ci, le ou les dictionnaires actifs avant que le programme EEA ne soit lancé, seront réactivés automatiquement.

Syntaxe
Chargerdico(dictionnaire)
Il n'y a pas de valeur retour.

L'expression argument de cette instruction est une chaîne de caractères qui doit être égale au nom du dictionnaire à charger (il est recommandé de compléter le nom par son chemin d'accès, selon les règles habituelles d'accès aux fichiers).

Exemples

```
Chargerdico("c:\\Program Files\\JLP Fractware\\Scrabble.ajl")
```

Si vous désirez charger plusieurs dictionnaires à la fois, il suffit de séparer leurs noms par un **point virgule**.

```
Chargerdico("c:\\Program Files\\JLP Fractware\\Scrabble.ajl;c:\\Program Files\
\JLP Fractware\\Monbeaudico.txt")
```

Le plus souvent, cette instruction précède une boucle Liredico - Finliredico, et son argument est obtenu dynamiquement par appel à le fonction Saisiefichier

Avertissement
Il n'est pas autorisé de changer de dictionnaires "à la volée" en cours de lecture dans un Bloc Liredico - Finliredico, que ce soit par appel à Chargerdico ou encore par le menu Fichier. Ainsi, cette séquence d'instruction amène à coup sûr un plantage du programme AJL :

```
Chargerdico(nouveaudico1)
Liredico(mot)
  ... instructions...
  Chargerdico(nouveaudico2)
Finliredico
```

En revanche, il est autorisé de changer de dictionnaire après sa lecture complète. Ainsi, cette séquence est

autorisée :

Chargerdico(nouveaudico1)
Liredico(mot)
  ... instructions...
Finliredico

Chargerdico(nouveaudico2)
Liredico(mot)
   ... instructions...
Finliredico

Voir aussi
Bloc Liredico - Finliredico
Saisiefichier

Debutmot

Debutmot(chaine)

La fonction **Debutmot** teste si une chaîne de caractères est un début d'un mot du dictionnaire.

Syntaxe
Debutmot(argument)
Le résultat est de type booléen, argument est de type chaîne.

Exemples
Debutmot("RTFM")
Avec le dictionnaire standard, le résultat est FAUX, donc la valeur entière 0.

Debutmot("DIC")
Avec le dictionnaire standard, le résultat est VRAI car au moins un mot commence par "DIC".

Dicoliste

Dicoliste(liste)

La fonction **Dicoliste** crée une liste contenant chacun des mots du dictionnaire

Syntaxe
Dicoliste(liste)
La fonction ne renvoie pas de résultat, l'argument liste est mis à jour (ou créé) avec le contenu du dictionnaire en cours

Exemple
Dicoliste(D)

Lexique

Mot

Mot(chaine)

La fonction **Mot** teste si une chaîne de caractères est un mot du dictionnaire.

Syntaxe
Mot(argument)
Le résultat est de type booléen, argument est de type chaîne.

Exemple
Mot("BLEU")
Avec le dictionnaire standard, le résultat est VRAI donc la valeur entière 1.

Tiragemot

Tiragemot()
ou
Tiragemot(lgmin,lgmax)

La fonction **Tiragemot** tire un mot du dictionnaire au sort. Optionnellement, on peut fournir deux arguments pour ne sélectionner que les mots dont la longueur est comprise (bornes incluses) dans les limites de longueur spécifiées. Si aucun mot ne respecte les contraintes de longueur imposées, la valeur retournée est une chaîne vide.

Deux syntaxes possibles, avec zéro ou deux arguments
Tiragemot()
Le résultat est de type chaîne, il n'y a pas d'argument d'entrée.

Tiragemot(min,max)
Le résultat est de type chaîne, les deux arguments sont des entiers, respectivement longueur minimale et maximale souhaitées.

Exemples
Selection(Tiragemot())
Cette ligne de programme affiche un mot du dictionnaire au hasard.

Selection(Tiragemot(5,8))
Cette ligne de programme affiche un mot du dictionnaire tiré au hasard parmi ceux de longueur comprise (bornes incluses) entre 5 et 8.

Bibliothèque d'exemples

Bibliothèque d'exemples

Mode d'emploi de la bibliothèque
Utilisez cette bibliothèque d'exemples prêts à l'emploi pour découvrir les amusantes possibilités de EEA.
tout ce que vous avez à faire, c'est :
1) copier-coller le contenu d'un exemple dans la fenêtre principale de l'onglet EEA (pour copier :

sélectionnez à la souris le texte contenu dans le cadre sans déborder hors du cadre, faites clic-droit et choisissez "copier" dans le menu qui surgit. Puis dans AJL ouvert à l'onglet EEA faites Edition->Coller ou simplement CTRL+V).
2) lancer le programme EEA en appuyant sur OK !

Le niveau de difficulté de ces programmes est évalué :

* = court et facile, une minute suffit pour piger le fonctionnement

** = pas de difficulté, mais suffisamment long pour y consacrer quelques minutes

*** = pas facile d'accès, il faut se concentrer un peu pour tout bien piger.

**** = si vous trouvez comment çà marche, vous serez gentils de me faire un mail.

Vous pouvez aussi utiliser le menu Fichier -> Ouvrir script EEA et lancer un des **nombreux scripts prêts à l'emploi livrés dans l'installation de AJL**.

Prise de contact EEA *
Une rapide découverte des instructions de base

Les septs lettres revenentes *
Un hommage à Georges Perec

Compter les lettres *
Combien de lettres dans le dictionnaire ?

Trop plein de consonnes ***
Ce script cherche les mots bourrés de consonnes

Mots isocèles **
Ce script cherche les mots isocèles. Ah ! ca vous épate, hein ?

Les mots imbriqués **
Ce script recherche les paires de mots cachées dans un mot

Le Morse c'est nul ***
Afin de démontrer que le code Morse est nul.

Un épeleur de nombres ***
Construit une suite logique de nombres simplement en les épelant chiffre à chiffre

La Sextine voyelles **
Un hommage à Raymond Queneau

Les pires mots qui soient **
Recherche les mots qui contiennent un grand nombre de fois la même lettre

Additionner des mots **
Saviez-vous qu'on peut additionner certains mots ?

La densité d'un dictionnaire **
Combien ca pèse, en moyenne, un dictionnaire? hein ?

Fonction Nombre en lettres ***
ce n'est pas un script mais une fonction outil capable d'écrire en toutes lettres n'importe quel chiffre.

Cette phrase compte trente lettres *

Un exemple d'utilisation de la fonction précédente. Il y a vraiment 30 lettres dans "cette phrase compte trente lettres" !

La transformation de Cambridge ****

Chamboulez les lettres d'un texte tout en le laissant parfaitement lisible !

S+7 ****

Une implémentation de la célèbre transformation "S+7" chère à l'OULIPO.

Découpe mots ****

Cherche les mots découpables en plus petits mots, comme RAT-TACHE-MENT = RATTACHEMENT

Démonstration multi-tâches **

Un programme de démonstration de mise en œuvre de la programmation parallèle multi-tâches

Extracteur de mots ***

Extrait les mots d'un texte.

Les nombres de Schulz **

Script ultra compact qui trouve l'ensemble des nombres de Schulz. Technique de programmation en liste.

Résolution du sudoku *****

Article très technique, consacré à l'application de la programmation en liste à la résolution des sudokus.

Prise de contact EEA

```
//==============================================================
// Prise de contact avec EEA.
//==============================================================
Messagebox("Bienvenue dans le monde de EEA. Appuyez sur OK pour continuer")
Messagebox("L'instruction Message permet d'afficher dans la zone message en
bas à gauche.")
Message("Coucou! ceci est un message !")
dix = 5 + 5
Message(dix)
xx = 2*4 + 3*3 + 3
Message("vi" + "ngt" + " = " + xx)

Messagebox("L'instruction Selection permet d'afficher dans la sélection à
droite.")
Selection ("ceci est un mot")
zzz = "autre mot"
Selection("ceci est un " + zzz)

Messagebox("Le bloc d'instruction boucle - finboucle permet de faire des
boucles")
boucle(i=1; i <=10 ; i = i + 1)
  Message("i = "+i)
finboucle
```

```
Messagebox("Le bloc d'instruction si - sinon - finsi permet de faire des
tests")
boucle(i=1; i <=10 ; i = i + 1)
  si (i <= 5)
    Message(i+" est petit")
  sinon
    Selection(i+" est grand")
  finsi
finboucle

Messagebox("Le bloc liredico - finliredico permet de parcourir le
dictionnaire")
Messagebox("Appuyez sur le bouton Interrompre pour stopper le programme, ou
observez l'arrêt automatique au 3000ème mot")
i = 0
liredico(mot)
  i += 1
  Message(mot)
  si (i >= 3000)
    arretliredico
  finsi
finliredico
```


Les septs lettres revenentes

```
//===============================================================
// En pensée de G. Perec, "Les Sept Lettres Revenentes"
// (Référence célèbre) cherche les textes de sept lettres
// exempts de v*yelles excepté "E", et les révèle.
//===============================================================
// (c) JLP Frectwère. En trente-sept + cent sept + sept*(cent trente + cent
trente sept)
//===============================================================

liredicoex(mot;"";7;7)

  si (mot $& "AIOUY" == "") //notez l'emploi de l'opérateur intersection $&
    Selection(mot)
  finsi

finliredico
```


Compter les lettres

```
// Ce programme compte le nombre d'apparitions
// de chaque lettre dans le dictionnaire.
// Il illustre l'utilisation des tableaux et le
// fait que les indices de tableaux peuvent être
// des lettres.
//
// (JLP - 01/2013)

Boucle(i=1;i<=26;i+=1)
  n[Lettre(i)] = 0
Finboucle
```

```
n = 0

LIREDICO(mot)
  Boucle(i=1;i<=Longueur(mot);i+=1)
    n[mot::i] += 1
    n += 1
  Finboucle
FINLIREDICO

Boucle(i=1;i<=26;i+=1)
  Selection("Nombre de "+Lettre(i)+" = "+n[Lettre(i)])
Finboucle
Selection("Nombre total de lettres : "+n)
```


Trop plein de consonnes

```
//=================================================================
// Ce programme cherche les mots ayant au moins 4 consonnes
// consécutives. Il donne des exemples.
//=================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//=================================================================
nconsonnes = 4  //nombre minimal de consonnes souhaité

n=0
c = Majus("bcdfghjklmnpqrstvwxz")

liredico(mot)
  l = Longueur(mot)
  i = 0
  tantque(i<l)
    x = Debutinclus(mot<<i,c)
    si (x >= nconsonnes)
      consonnes = Partie(mot,i+1,x)
      si(Existe(nbre[consonnes]))
        nbre[consonnes] += 1
      sinon
        nbre[consonnes] = 1
        consonnes[n] = consonnes
        mots[consonnes] = mot
        n += 1
      finsi
      i += x
    sinon
      i += 1
    finsi
  fintantque
finliredico

for(i=0;i<n;i+=1)
  Selection(nbre[consonnes[i]]+" mots en "+consonnes[i]+", exemple : 
"+mots[consonnes[i]])
endfor
```

Mots isocèles

```
//=====================================================
// Ce programme recherche les mots "absolument
// isocéles", c'est à dire les mots tels que la
// différence entre deux lettres consécutives est
// toujours la même, en valeur absolue.
// Un mot isocèle parfait ne comporte que des
// lettres toutes différentes
//=====================================================
// (JLP 06/01/2013)
//=====================================================

Liredico(mot)
  l=Longueur(mot)
  SI(l > 2)
    c = Rang(mot::2)-Rang(mot::1)
    c = Max(c,-c)//prends la valeur absolue

    Boucle(i=2;i<l;i+=1)
      d = Rang(mot::(i+1))-Rang(mot::i)
      d = Max(d,-d)//prends la valeur absolue
      si(c!=d)
        suiteliredico
      finsi
    Finboucle

    si (Longueur(mot $& "ABCDEFGHIJKLMNOPQRSTUVWXYZ") == l)
      Selection(mot+" isocèle parfait côté=" + c)
    sinon
      Selection(mot+" isocèle, côté=" + c)
    finsi

  FINSI
Finliredico
```


Les mots imbriqués

```
//====================================================================
//ce script recherche les mots d'au moins 8 lettres dont les lettres
//de rang pair et impair forment 2 autres mots
// exemple : EMPAILLEES  (MALES et EPILE)
//====================================================================
//script initial Hervé Bourgeois 15/7/2001 optimisation JLP mai 2012
//====================================================================

LIREDICOex(mot;"";8;99)
  l = Longueur(mot)

//on va construire mot[0] et mot[1] en alternant les lettres de mot

  mot[0] = ""
  mot[1] = ""

  boucle(m=1; m<=l; m+=1)
    mot[m%2] += mot::m
```

```
  finboucle

  si (Mot(mot[0]) && Mot(mot[1])) //les mots "imbriqués" sont valides
    Selection(mot+"  ("+mot[0]+" et "+mot[1]+")")
  finsi

FINLIREDICO
```


Le Morse c'est nul

```
//===========================================================================
// Ce script transcrit les mots du dictionnaire en morse et cherche les
// mots de même code morse.
// - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
// Le programme sélectionne les paires de mots d'au moins 4 lettres ayant
// même code morse, de même longueur et n'ayant aucune lettre en commun.
//
// Puis le programme indique en zone message :
// 1) le nombre de mots ayant le même code morse.
// 2) les deux mots de même code morse présentant le plus de différences
// 3) la liste des mots dont le code morse est commun avec au moins
//    9 autres mots.
// Conclusion : le morse est un code exceptionnellement mauvais.
//===========================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//===========================================================================

m["A"]=".-"
m["B"]="-..."
m["C"]="-.-."
m["D"]="-.."
m["E"]="."
m["F"]="..-."
m["G"]="--."
m["H"]="...."
m["I"]=".."
m["J"]=".---"
m["K"]="-.-"
m["L"]=".-.."
m["M"]="--"
m["N"]="-."
m["O"]="---"
m["P"]=".--."
m["Q"]="--.-"
m["R"]=".-."
m["S"]="..."
m["T"]="-"
m["U"]="..-"
m["V"]="...-"
m["W"]=".--"
m["X"]="-..-"
m["Y"]="-.--"
m["Z"]="--.."
dm=0
ii=1
nn=0
liredico(mot)
```

```
  l = Longueur(mot)

  //on construit le code morse dans la variable code
  for(i,code = 1,""; i <=l; i,code += 1, m[mot::i])
  endfor

  //on teste si ce code morse a déjà été rencontré
  si (Existe(c[code]))
    nn += 1 //compteur de mots ayant même morse
    n[code] += 1
    //évalue la différence entre le mot en cours et le 1er mot de même morse
    d = Min(Difference(c[code],mot),Difference(-c[code],-mot))
    //si c'est mieux, on conserve le record dans les variables r1 et r2
    si (d > dm)
      dm = d
      r1 = mot
      r2 = c[code]
    finsi
    //on sélectionne les mots à afficher : aucune lettre en commun ($&)
    //longueurs des mots supérieure à 3, longueurs identiques
    lc = Longueur(c[code])
    si (Min(lc,l) > 3 && (c[code] $& mot) == "" && lc == l)
      Selection(mot+" et "+c[code])
    finsi

  //sinon, c'est la 1ère fois qu'on le voit. On le stocke en tableau
  sinon
    c[code]= mot
    n[code] = 1
    t[ii] = code
    ii += 1
  finsi
finliredico

Message("mots de même code morse : %d",nn)
Message("mots de même code et de plus grande différence: %s et %s",r1,r2)
Message("liste des mots ayant plus de 9 doublons en code morse:")
for(j=1; j<ii; j+=1)
  a = t[j]
  if (n[a] > 9)
    Message(n[a]+" mots de même code que "+c[a])
  endif
endfor
```


Un épeleur de nombres

```
//===================================================================
// Ce script génère une suite de nombres dont la règle de production
// est la suivante : la premier élément est 1.
// Tout élément suivant est la forme épelée chiffre par chiffre du
// nombre précédent, ré-écrite sous forme numérique.
// exemple :
// 1
// 11 (le nombre précédent "1" se lit "un 1", réécrit sous forme "11")
// 21 (le nombre précédent "11" se lit "deux 1", réécrit 21 )
// 1211 (nombre précédent "21" se lit "un 2, un 1" réécrit 1211)
```

```
// etc...
// La longueur de chaque nombre est affichée devant le nombre.
//===================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//===================================================================

m = Entier(Saisie("Nombre d'éléments (pas plus de 20): "))
if (m > 20)
  Message("C'est trop, le nombre de doit pas être plus grand que 20")
  exit
endif

x = "1"
for(n = 1; n <= m; n += 1)
  l = Longueur(x)
  Selection(Format("lg = %3i :  %s",l,x))
  w = x
  for(x = ""; Longueur(w) > 0; x += i + a)
    a = w::1 //a = premier chiffre de w
    i = Debutinclus(w,a) //i = combien de fois à la suite voit-on ce
chiffre?
    w = w<<i // élimine les i premiers chiffres identiques
  endfor
endfor
```


La sextine voyelles

```
//==================================================================
// Ce programme cherche les mots qui donnent un autre mot après une
// ou plusieurs permutations des voyelles selon la sextine chère
// à Raymond Queneau. Pour une explication plus détaillée de la
// sextine, relire les ouvres complètes de Queneau.
// Sinon, sachez qu'une sextine appliquée sur "AEIOUY" donne
// successivement YAUEOI, IYOAEU, UIEYAO, OUAIYE, EOYUIA
// puis revient à AEIOUY.
//
// On peut modifier les lettres utilisées pour
// cette permutation en changeant le contenu de la
// variable "voyelles". Cependant ce nombre doit
// rester égal à 6.
//
// Le test sur "DIFFERENCE" permet d'éliminer de la
// sélection les mots qui contiennent moins de 2
// des voyelles permutées
//==================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//==================================================================

voyelles="AEIOUY"

SI(Longueur(voyelles)!=6)
  Message("Le nombre de lettres doit être 6 !")
  exit
FINSI
LIREDICO(mot)
  t=voyelles
  for(i = 1;  i < 6; i += 1) // effectue le chamboulement 6 fois de suite
```

```
    t = t::6 + t::1 + t::5 + t::2 + t::4 + t::3 //chamboule l'ordre des 6
voyelles
    new = Transpose(mot,voyelles,t) // calcule le mot avec ses voyelles
"chamboulées"
    SI(Mot(new) && Difference(mot,new) > 2) //est-ce un mot intéressant ?
      Selection(mot+" -> "+new+" ("+i+")")
    FINSI
  endfor
FINLIREDICO
```


Les pires mots qui soient

```
//======================================================================
//Ce script recherche les mots contenant le plus de fois possible la
//même lettre. il affiche le résultat en autant de lignes que de
//lettres différentes présentes dans le dictionnaire, avec pour
//chaque lettre le nombre maximal d'occurrences trouvées dans le mot
//record, et le mot record en question. En cas d'égalité du nombre
//de lettres contenues, le script ne retient que le mot le plus petit.
//======================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//======================================================================


for(i=1; i<=26; i+=1)
  RecordNbLettres[i] = 0  //initialiser le tableau des records à 0
endfor


liredico(mot)
  lg = Longueur(mot)
  for(i=1; i<=26; i+=1)
    NbLettres[i] = 0      //initialiser le décompte de lettres du mot
  endfor

  for(i=1; i<=lg; i+=1)  //l'indice i parcours le mot lettre à lettre
    NbLettres[Rang(mot::i)] += 1 //compte chaque lettre
  endfor

  for(i=1; i<=26; i+=1)
    //si le nbre d'occurence d'une lettre est plus grande que le record
    //actuel, mettre à jour le record
    if(NbLettres[i] > RecordNbLettres[i])
      RecordNbLettres[i] = NbLettres[i]
      MotRecord[i] = mot
    endif
    //si le nbre d'occurence d'une lettre est égal au record actuel
    //examiner les longueurs du record et du mot courant
    if(NbLettres[i] == RecordNbLettres[i])
      if (Existe(MotRecord[i])) //est-ce qu'un mot record existe déjà?
       //le nouveau mot est le nouveau record s'il est plus court
        if (lg < Longueur(MotRecord[i]))
          MotRecord[i] = mot
        endif
      else
      //le mot en cours est le tout premier mot record pour cette lettre
        MotRecord[i] = mot
```

```
      endif
    endif

  endfor
finliredico

//affichage des résultats sur 26 (ou moins) lignes
for(i=1; i<=26; i+=1)
  if (Existe(MotRecord[i]))
    Selection(RecordNbLettres[i]+Lettre(i)+" "+MotRecord[i])
  endif
endfor
```


Additionner les mots

```
//===================================================================
// Ce programme "additionne" deux mots
// de même longueur et contrôle si le
// résultat est un mot du dictionnaire.
//
// L'addition se fait lettre par
// lettre : le rang dans l'alphabet de
// chaque lettre est additionné, donnant
// le rang de la lettre résultat, modulo 26
// ex : A + B = C; Y + C = B.
// il n'y a pas de retenue.
//
// La longueur des mots sélectionnés est
// réglée par le paramètre lgm obtenu par SAISIE
//==================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//==================================================================

lgm=Entier(Saisie("Longueur des mots à additionner ?"))

liredicoex(mot1;"";lgm;lgm)
  liredicoex(mot2;mot1;lgm;lgm)//boucle dictionnaire imbriquée

    mot3="" //initialise le mot résultat à vide.
    for(i=1; i<= lgm; i += 1) //explore toutes les lettres
      mot3 += (mot1::i) ++ (Rang(mot2::i)) //calcule la "somme" lettre à
lettre
    endfor

    if(Mot(mot3)) //teste si le mot résultat existe
      Selection(mot1+"+"+mot2+"="+mot3)
    endif

  finliredico
finliredico
```


La densité d'un dictionnaire

```
//==================================================================
// Ce programme calcule la distribution de densité du
```

```
// dictionnaire. Le poids d'un mot est défini par
// la somme des rangs de chacune de ses lettres. Sa
// densité est son poids divisé par sa longueur.
// La distribution de densité est le nombre de mots
// par intervalle de densité de longueur 1.
// Ca l l'air compliqué comme çà, mais en faisant tourner
// ce petit programme tout s'éclaire en un instant.
//==================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//==================================================================

//initialisation à zéro du tableau des densités.
for (i=0;i<27;i+=1)
  cumul[i] = 0
endfor

//valeur d'une lettre = son rang
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26)

//boucle dictionnaire
LIREDICO(mot)
  densite = Valeur(mot)/Longueur(mot)
  cumul[densite] += 1
FINLIREDICO

//fin de boucle dico : affichage des résultats
for(i=0;i<27;i+=1)
  SI (cumul[i] > 0)
    Selection("nombre de mots de densité "+i+" = "+cumul[i])
  FINSI
endfor
```


Fonction nombre en lettres

Mode d'emploi

Pour utiliser cette fonction, faites un copier- coller dans EEA et enregistrez sous le nom "Nel.eea" dans le même répertoire que celui où vous avez installé AJL.exe

```
//==================================================================
// Fonction Nel : Nombre En Lettres
// Cette fonction transcrit un nombre entier inférieur à 2147483647
// sous sa forme littérale, selon les règles de l'orthographe à la
// française (les formes belges et suisses ne sont pas gérées).
// Toutefois l'adaptation à ces formes ne présente pas de difficulté.
// Il suffit de mettre à jour les tableaux
// IndexDiz, IndexUnit, et Dizaine aux valeurs 7 pour septante
// 8 pour octante et 9 pour nonante, selon les usages voulus.
//==================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//==================================================================


%Nel(nbre)

if (nbre <= -2147483647 || nbre >= 2147483647)
  result = "EEA n'sait point compter jusque là."
```

```
  exit
endif
if (nbre < 0)
  result = "moins " + Nel(-nbre)
  exit
endif

Chiffre[0] = ""
Chiffre[1] = "un"
Chiffre[2] = "deux"
Chiffre[3] = "trois"
Chiffre[4] = "quatre"
Chiffre[5] = "cinq"
Chiffre[6] = "six"
Chiffre[7] = "sept"
Chiffre[8] = "huit"
Chiffre[9] = "neuf"
Chiffre[10] = "dix"
Chiffre[11] = "onze"
Chiffre[12] = "douze"
Chiffre[13] = "treize"
Chiffre[14] = "quatorze"
Chiffre[15] = "quinze"
Chiffre[16] = "seize"
Chiffre[17] = "dix-sept"
Chiffre[18] = "dix-huit"
Chiffre[19] = "dix-neuf"

if (nbre <= 19)
  result = Chiffre[nbre]
  exit
endif

Dizaine[2]  = "vingt"
Dizaine[3]  = "trente"
Dizaine[4]  = "quarante"
Dizaine[5]  = "cinquante"
Dizaine[6]  = "soixante"
Dizaine[8]  = "quatre-vingt"

IndexDiz[2] = 2
IndexDiz[3] = 3
IndexDiz[4] = 4
IndexDiz[5] = 5
IndexDiz[6] = 6
IndexDiz[7] = 6
IndexDiz[8] = 8
IndexDiz[9] = 8

IndexUnit[2] = 0
IndexUnit[3] = 0
IndexUnit[4] = 0
IndexUnit[5] = 0
IndexUnit[6] = 0
IndexUnit[7] = 10
IndexUnit[8] = 0
IndexUnit[9] = 10
```

```
Cet[0] = " "
Cet[1] = " et "
Pluriel[0] = ""
Pluriel[1] = "s"
result = ""

if (nbre <= 99)
  s = Pluriel[nbre == 80]
  cet = Cet[nbre%10 == 1 && nbre <= 71]
  result = Dizaine[IndexDiz[nbre/10]] + s + cet + Nel(nbre%10
+IndexUnit[nbre/10])
  exit
endif

if (nbre <= 999)
  if (nbre/100 == 1)
    result = "cent" + " "+ Nel(nbre%100)
  else
    s = Pluriel[nbre%100 == 0]
    result = Chiffre[nbre/100] + " " + "cent" + s + " " + Nel(nbre%100)
  endif
  exit
endif

Puissance[1]= "mille"
Puissance[2]= "million"
Puissance[3]= "milliard"

if (nbre <= 1999)
  result = Puissance[1] + " " + Nel(nbre%1000)
else
  d = (Longueur(nbre) - 1)/3
  n = 10^(d*3)
  s = Pluriel[!((d == 1 || (nbre/n) == 1))]
  result = Nel(nbre/n) + " " + Puissance[d] + s + " " + Nel(nbre%n)
endif

%return(result)
//==========================================================
```


Cette phrase compte trente lettres

```
//=========================================================================
// Ce programme calcule 100 phrases dans la variable Test selon le modèle
// "Cette phase compte une lettres"
// "Cette phase compte deux lettres"
// ... etc ...
// "Cette phase compte cent lettres"
//
// Puis il calcule la variable TestSansEspaces qui est la phrase Test
// de laquelle on a ôté tous les espaces et tirets, comme par exemple
// "Cettephrasecomptedixseptlettres"
//
// Enfin, il calcule la longueur de cette dernière phrase, et donc
// le nombre de lettres. Si sa longueur est le même nombre que le
// chiffre "en lettres" utilisé, le programme affiche sa trouvaille.
// Vous verrez qu'il n'y a qu'une seule solution selon ce modèle.
```

```
//==========================================================================
// (c) JLP Fractware avril 2012  -   Pour AJL version 5.
//==========================================================================
//il faut que Nel.eea soit dans le même répertoire que AJL.exe
//pour celà, il faut d'abord copier- coller dans EEA l'exemple précédent
intitulé
//"Fonction nombre en lettres", et l'enregistrer sous le nom "Nel.eea" dans
//le même répertoire que celui où vous avez installé AJL.exe

#include(Nel.eea)

for(n = 1; n <=100; n += 1)

  Test = "Cette phrase compte "+Nel(n)+" lettres"
  TestSansEspaces = Test - " -" //l'opérateur - retire espaces et tirets

  if (Longueur(TestSansEspaces) == n)
    Selection(Test)
  endif
endfor
```


La transformation de Cambridge

```
//-------------------------------------------------------------------
// Ce script applique la transformation de "Cambridge" à un texte.
// Cette transformation consiste à mélanger les lettres des mots
// de plus de trois lettres, à l'exception de la première et de la
// dernière lettre, le truc étant que le texte est censé rester
// lisible. La technique de mélange adoptée ici consiste à tirer
// au sort deux lettres du mot (sauf la 1ère et la dernière) et à
// les échanger. L'échange est réalisé deux fois au moins en
// vérifiant qu'on obtient un mot différent du mot initial.
// Au choix de l'utilisateur, les mots sont écrits dans un fichier
// ou présentés dans la fenêtre SELECTION.
//
// JLP (01/2013)
//-------------------------------------------------------------------

//--- paramètres modifiables ---------
NbMelange = 2      //nombre de fois que l'on échange des lettres entre elles
//--------------------------------------

%Cambridge(mot1,NbMelange)
    mot2= mot1 //mot2 sera le mot après transformation
    Si(Longueur(_alphabet $& (mot1<<1)>>1) >= 2) //cf (*1) en fin de script
      lm = Dissocie(xm,mot1) //découpe mot en lettres dans xm
      Tantque(mot2 == mot1) //vérifie que le mot a changé
      Boucle(i=1; i<=NbMelange ;i+=1) //fait plusieurs échanges
        j = 1+Hasard(lm-2) //indice d'une lettre à échanger
        k = 1+Hasard(lm-2) //autre indice
        xm[j],xm[k] = xm[k],xm[j] //échange les 2 lettres //cf(*2)
      Finboucle
      mot2 = Associe(xm,lm) //reconstruit le mot depuis le tableau xm
      Fintantque
    Finsi
%return(mot2)
```

```
_alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzàâäçéèêëîïôöùûüÿÀÂÄÇÉÈÊ
ËÎÏÔÖÙÛÜ"
f = Saisiefichier("Nom du fichier à transformer ?")
o = Saisiefichier("Nom de fichier en sortie ? (annuler = sélection)")
LIREFICHIER(f;l1) //l1 est la ligne d'entrée
  l2 = ""  //l2 est la ligne de sortie
  x = Longueur(l1)
  Tantque(x != 0)
    a = Debutnoninclus(l1,_alphabet)  //partie sans lettres
    l2 += Debut(l1,a)  //transférée sans changement dans l2
    l1 = l1<<a  //et éliminée de la ligne d'entrée l1
    b = Debutinclus(l1,_alphabet) //partie en lettres seules
    l2 += Cambridge(Debut(l1,b),NbMelange)//transformation !
    l1 = l1<<b //élimine le mot traité de la ligne d'entrée
    x = Longueur(l1)
  Fintantque
  Si(o != "")
    Ecrirefichier(o,l2)
  Sinon
    Selection(l2)
  Finsi
FINLIREFICHIER

//-------------------------------------------------------------------
// Nota (*1) sur l'instruction complexe de la ligne 22.
//-------------------------------------------------------------------
//Cette instruction complexe vise à éviter les mots
//d'au moins 3 lettres qui à l'évidence ne peuvent être "Cambridgés"
//ou les mots comme "ELLE" où l'échange des 2 lettres centrales "L"
//entre elles ne change pas le mot.
//Le mot est d'abord débarrassé de sa 1ère et dernière lettre, ce
//qui est obtenu avec (mot<<1)>>1.
//Puis le sous ensemble des lettres de l'alphabet necessaires
//pour construire le mot est obtenu par intersection entre
//l'alphabet et le mot : alphabet $& (mot<<1)>>1.
//Enfin, la longueur de ce sous-ensemble donne le nombre de lettres
//différentes utilisées dans le mot (sauf 1ère et dernière lettre).
//On peut "Cambridger" s'il y a au moins 2 lettres différentes.
//-------------------------------------------------------------------
// Nota (*2) sur l'instruction de la ligne 28.
//-------------------------------------------------------------------
//l'utilisation d'une affectation de liste à deux éléments comme
// a,b = b,a         permet d'échanger deux éléments entre eux
//sans recourir à la forme classique à 3 instructions :
// x = a
// a = b
// b = x
//--------------------------------------------------------------------
```


S+7

```
//-------------------------------------------------------------------
// Ce script remplace les mots d'un texte, de 4 lettres ou plus, qui
// sont présents au dictionnaire courant, par le mot situé 7 rangs
// plus loin dans ce dictionnaire.
```

```
// JLP (01/2013)
//-------------------------------------------------------------------

%PlusSept(mot1)
  mot2 = mot1
  Si (Longueur(mot1) > 4)
    i = 0
    LiredicoEx(mot2;Majus(mot1);1;99)
      si (i==0 && mot2 != Majus(mot1))
        //le mot n'a pas été trouvé. on ne change pas le mot
        mot2 = mot1
        arretliredico
      finsi
      i += 1
      si (i == 7)
        //on change le mot par le 7ème plus loin.
        mot2 = Minus(mot2)
        arretliredico
      finsi
    Finliredico
  Finsi
%return(mot2)

alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzàâäçéèêëîïôöùûüÿÀÂÄÇÉÈÊ
ËÎÏÔÖÙÛÜ"
f = Saisiefichier("Nom du fichier à lire ?")
o = Saisiefichier("Nom de fichier en sortie ? (annuler = sélection)")
n = 0
LIREFICHIER(f;l1) //l1 est la ligne d'entrée
  l2 = ""  //l2 est la ligne de sortie
  Tantque(Longueur(l1) != 0)
    a = Debutnoninclus(l1,alphabet)  //isole partie sans lettres
    l2 += Debut(l1,a)  //transférée sans changement dans l2
    l1 = l1<<a  //et éliminée de la ligne d'entrée l1
    b = Debutinclus(l1,alphabet) //isole partie en lettres seules
    l2 += PlusSept(Debut(l1,b)) //transfère mot+7 dans la sortie
    l1 = l1<<b //élimine le mot traité de la ligne d'entrée
  Fintantque
  Si(o != "")
    Ecrirefichier(o,l2)
  Sinon
    Selection(l2)
  Finsi
FINLIREFICHIER
```


Découpe mots

```
//=====================================================================
//-------------- Sur une idée de ALAIN ZALMANSKI ---------------------
//=====================================================================
//Un bon exemple de la puissance de la programmation récursive.
//Ce script recherche les mots qui peuvent se découper en une
//suite de mots. Comme par exemple AVANTAGEUSE = AVANT-AGE-USE
//il écrit les mots trouvés dans un fichier dont le nom est
//demandé en début de programme. Surtout, indiquez bien un nom
//de fichier texte avec extension 'txt'. Exemple:  decoupemots.txt
```

```
//Pour limiter le nombre (énorme) de solutions écrites,
//le programme ne retient que les mots découpés en au moins trois
//morceaux, et dont chaque morceau comporte au moins trois lettres.
//Dans ces conditions et avec le dico scrabble, il y a 42096 solutions.
//Pour changer ces paramètres, voir les commentaires avec "<====".
//====================================================================
// Pour AJL version 5.
//====================================================================
%decoupemot(solution,mot,nbre)
l = Longueur(mot)
if (l == 0) //si mot = vide, c'est qu'on a une solution
  if (nbre >= 3 ) // <==== exige au moins 3 morceaux
    Ecrirefichier(_fic,solution<<1) //<<1 élimine le 1er "-" inutile
    _nsol += 1 //notez les variables globales _nsol et _fic
  endif
  exit //fin de fonction car la solution a été traitée.
endif

//sinon, on continue à découper par appels récursifs
for(i=3; i<=l; i+=1)// <==== au moins 3 lettres pour un morceau
  if(Mot(Debut(mot,i))) //essaye ce découpage.
    decoupemot(solution+"-"+Debut(mot,i), mot<<i, nbre+1)
  endif
endfor
%return()

_fic = Saisiefichier("Donnez le nom du fichier à créer")
if (_fic == "")
  exit
endif

_nsol = 0
Liredico(mot)
  decoupemot("", mot, 0)
Finliredico
Message("%d solutions écrites dans le fichier",_nsol)
```


Extracteur de mots

```
//------------------------------------------------------------------
// extracteur des mots d'un texte
// JLP (01/2013)
//------------------------------------------------------------------
alphabet =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyzàâäçéèêëîïôöùûüÿÀÂÄÇÉÈÊ
ËÎÏÔÖÙÛÜ"

f = Saisiefichier("Nom du fichier texte à lire ?")
si (f == "")
  exit
finsi

o = Saisiefichier("Nom de fichier de mots à créer ? (annuler = affichage
seul)")
nmots = 0

LIREFICHIER(f;l1) //l1 est la ligne d'entrée
```

```
  x = Longueur(l1)
  Tantque(x != 0)
    l1 = l1<<Debutnoninclus(l1,alphabet)
    b = Debutinclus(l1,alphabet) //partie en lettres seules
    mot = Debut(l1,b)
    si (!Existe(TMot[mot]) && mot != "")
      nmots += 1
      TMot[mot] = nmots
      Index[nmots] = mot
    finsi
    l1 = l1<<b //élimine le mot traité de la ligne d'entrée
    x = Longueur(l1)
  Fintantque
FINLIREFICHIER

Trialpha(Index,1,nmots)
for(i=1;i<=nmots;i+=1)
  if (o == "")
    Selection(Index[i])
  else
    Ecrirefichier(o,Index[i])
  endif
endfor
```


Démonstration multi-tâches

```
//=====================================================
//     programme de démonstration du multi-thread
//=====================================================
//le programme principal appelle deux fois la même
//fonction "calculxy". Cette fonction calculxy
//appelle une fonction "calcul" pour toutes les valeurs
//des nombres entre x et y.
//si vous conservez le mot clé "thread" de la fonction
//calculxy, vous constatez que les calculs sur les séries
//de nombres 1-500 et 501-1000 s'effectuent en parallèle
//en un temps réduit. Si vous ôtez ce mot clé, vous
//constatez que les deux séries sont traitées l'une
//après l'autre, en un temps supérieur.
//=====================================================

//fonction calcul sur une valeur x.
%calcul(x)
%return(Hasard(x))

//fonction pour réaliser le calcul sur tous les nombres de x à y
%calculxy(x,y) thread
n = 0
for(i = x; i <= y; i += 1)
n += calcul(i)
Selection("Calcul(%d) effectué",i)
endfor
%return(n)

//programme principal : deux appels de calculxy
Message("Total = %d",calculxy(1,500)+calculxy(501,1000))
```

Les nombres de Schulz

```
//=========================================================
// Ce script recherche les nombres de Schulz, c'est-à-dire
// les nombres dont la somme des rangs des lettres de leur
// forme littérale est le nombre lui même. Selon cette
// définition stricte, il n'en existe que 3 : 222, 232, 258
//=========================================================
#include(Nel.eea)
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,22,23,24,25,26)
N = %500 // tester les nombres de 1 à 500

Selection(N & °Valeur(°Majus(°Nel(N)))  °== N)
```

Explication du programme :

`N` = %500 est la liste des nombres de 1 à 500 : {1,2,3,...,500}

`°Nel(N)` est la liste des formes littérales de ces nombres : {"un","deux",...,"cinq cents"}. Il est fait appel à la fonction Nel.eea fournie avec AJL. ° est l'opérateur de distribution.

`°Majus(°Nel(N))` est donc la version majuscule de cette liste : {"UN","DEUX",...,"CINQ CENTS"}

`°Valeur(°Majus(°Nel(N)))` est la liste des valeurs de la liste précédente : la valeur d'un mot est la somme des valeurs de ses lettres, et la valeur de chaque lettre a été fixée à son rang dans l'alphabet par la fonction `Poids` en début de programme. "A" = 1, "B" = 2, etc... "Z" = 26. Ce qui fait que "UN" donne 21 + 14 = 35. Le résultat global est donc {35,54,...,104}

`°Valeur(°Majus(°Nel(N)))  °== N` compare la liste des valeurs obtenues avec la liste des nombres initiaux {1,2,...}. En cas d'égalité on obtient un 1, et 0 sinon. Le liste obtenue comporte 497 zéros et 3 uns, aux positions de rang 222, 232, 258, car la Valeur de "DEUX CENT VINGT-DEUX" est 222, et de même pour 232 et 258.

`N & °Valeur(°Majus(°Nel(N)))  °== N` sélectionne les éléments de la liste initiale N, correspondants aux 1 de la liste précédente, ce qui donne la liste de 3 éléments {222,232,258}

Résolution du sudoku

Résolution du jeu de SUDOKU
Une illustration de la programmation en liste

Pour ceux qui sont pressés de trouver la solution d'un Sudoku et se fichent de la façon de faire, vous trouverez, dans votre répertoire d'installation de AJL, un script appelé Sudoku.eea. Il suffit de modifier les numéros de la grille utilisée en exemple, et de faire "OK" !

Pour ceux qui sont plus curieux, et souhaitent découvrir un exemple assez complexe illustrant la programmation "en listes", suivez moi, attachez vos ceintures, et réservez-vous une heure ou deux, bien au calme.

Si dans ce texte vous rencontrez une fonction ou un opérateur qui vous sont inconnus, allez en bas du document, toutes les fonctions utilisées y sont répertoriées avec leur nom, une description courte, et un lien vers l'aide détaillée de AJL.

L'idée de base de la résolution d'un Sudoku est la suivante : partir de la grille initiale, et déterminer la liste des nombres candidats possibles à la "meilleure" case encore libre de la grille. En déduire une liste de nouveaux problèmes de Sudoku, qui ne diffèrent du problème initial que par le fait qu'une case de plus est remplie. Et continuer ce processus jusqu'à ce qu'il ne reste plus aucune case libre. C'est un algorithme d'attaque brute, plutôt stupide, qui nous servira d'illustration de la programmation "en listes". Ce type de programmation est, comme vous le verrez, très différente de la programmation structurée conventionnelle, basée sur des blocs itératifs while-endwhile ou for-endfor. Nous testerons ce programme sur l'auto-proclamé "sudoku le plus difficile du monde" (11.3 à l'évaluation standard SE, tout de même !!!)

Nous aurons donc besoin :
-a- de déterminer comment représenter une grille de Sudoku en langage EEA,
-b- d'un outil pratique permettant de gérer les règles du jeu de Sudoku (gestion des contraintes en lignes, colonnes, blocs),
-c- de déterminer la liste des nombres candidats à chaque position libre de la grille,
-d- de constituer des suites de grilles dérivées d'un grille "source" complétée, à une position libre bien choisie, des nombres candidats à cette position.

ELABORER LA SOLUTION PAS A PAS

Le point -a- pourrait être traité à l'aide d'une liste, comportant 81 éléments, les 9 premiers pour la 1ère ligne, et ainsi de suite ligne par ligne. Mais une telle liste de taille fixe ressemble furieusement à une chaîne de caractères. Autant adopter cette solution : une grille sera donc une chaîne de 81 caractères, ayant soit la valeur "." si la case est libre, soit la valeur "1" à "9" si elle est occupée.

le point -b- est un peu plus délicat, surtout que notre grille (point a) est non structurée en ligne et colonne : c'est une simple chaîne "plate", l'accès à la valeur d'un élément donné ne peut se faire que par son numéro d'ordre de 1 à 81. Il faut donc commencer par construire une fonction permettant, à partir de la seule donnée donnée du rang de l'élément (1 à 81), de restituer 3 chiffres pour les numéros de ligne, de colonne et de bloc de l'élément.

C'est assez simple, un peu d'arithmétique élémentaire à base de modulo 9 et 3 fera l'affaire. Comme on désire renvoyer 3 valeurs en retour (ligne, colonne, bloc), on va organiser ces retours dans ... une liste de 3 éléments. Ainsi la fonction Lcb ci-après accepte une valeur d'entrée correspondant au rang (1 à 81) d'une cellule, et renvoie en retour 3 valeurs (entre 0 et 8) correspondant aux numéros de ligne, colonne, bloc de la cellule. Notez que les valeurs de retour sont entre 0 et 8 au lieu de 1 et 9, pour la seule et unique raison qu'ajouter +1 aux 3 résultats n'a pas d'intérêt.

```
%Lcb(ix)
 q = (ix-1)/9
 m = (ix-1)%9
%return({q , m , (m/3) + 3*(q/3)})
```

par exemple,

Lcb(3) --> 0,2,0 (ligne 0, colonne 2, bloc 0. (case 3 en ligne 0, colonne 2, bloc 0)
Lcb (81) --> 7,7,7 (case 81, la dernière, en ligne 7, colonne 7, bloc 7)

A partir de celà, il devient très simple de savoir si deux éléments d'index i et j entrent en contention. Il le sont, si et seulement s'ils partagent une même ligne, ou une même colonne, ou un même bloc. Une comparaison entre deux listes résultats de Lcb fournit toute l'information. On utilise pour celà une distribution de l'opérateur de comparaison: °==

R = (Lcb(i) °== Lcb(j))
Si R possède un de ses 3 composants à 1, c'est que les cases i et j partagent au moins une ligne ou une colonne ou un bloc. Il suffit de faire un OU logique sur les 3 composantes pour obtenir le résultat souhaité. Pour celà, on va utiliser une réduction portant sur l'opérateur OU logique : ||°

```
%Cmap(i,j)
%return(||°(Lcb(i)°==Lcb(j)))
```

Cmap(x,y) renvoie donc 1 si les deux cellules de rang x et de rang y sont en contention, 0 sinon. On peut précalculer tout celà, puisque le fait que deux cellules soient en contention ne dépend pas de leurs valeurs. Du coup, on peut imaginer un GRAND tableau de 81 lignes et 81 colonnes. Un élément de ce tableau, aux coordonnées ligne L et colonne C, aura la valeur 1 si l'élément de grille de rang L entre en contention avec l'élément de grille de rang C. Un produit externe exécutant notre fonction Cmap sur deux listes des nombres de 1 à 81 est exactement ce qu'il nous faut pour celà :

MAP1 = °Cmap°(%81,%81)

MAP1 est une liste plate de 81 x 81 = 6561 éléments, valant 0 ou 1. Seul problème, c'est UNE seule liste plate, alors qu'il nous faut 81 listes de 81 éléments, de sorte que chacune des 81 lignes recense toutes les cases en contention avec LA case représentée par cette ligne. Pas de problème, on va faire des paquets de 81 avec l'opérateur µ

MAP2 = MAP1 µ 81

Ainsi, la première grille de cette liste MAP2::1 = {..} contient 54 zéros et 27 chiffres 1 signifiant que les 27 cases des ces rangs sont en contention avec la case en position 1 dans la grillle. Idem avec la xème grille MAP2::x qui repère les 27 rangs des cases de la grille , en contention avec la case de rang x.
Ce serait tout de même plus commode d'avoir directement les 27 rangs où il y a des 1, plutôt que 81 nombres à 0 ou 1. Pas de souci . Il suffit d'exécuter une recherche L?1 sur chacune des 81 sous listes L de MAP2. Une distribution °? fera l'affaire.

CMAP = MAP2 °? 1

Au final :

```
CMAP = (°Cmap°(%81,%81) µ 81) °? 1
```

Le tout en une seule ligne, efficace non ? d:o-)

Passons au point -c-

Repérer les rangs des positions libres d'une grille revient à y chercher les "." :
L = S?"."
C'est la liste des positions libres.
Les rangs des éléments en contention avec cette position sont, directement (merci, CMAP !)
R = CMAP::L
Et les nombres de la grille S déjà présents à ces positions sont
N = S°::R
En conséquence, chacun de ces nombres est interdit, pour la position libre considérée. Quels nombres restent disponibles ? Il suffit de retirer chacun de ces nombres interdits de la liste des nombres de 1 à 9.
P = "123456789" °- N

Au final, si on résume tout çà, la liste des candidats possibles pour chaque case libre de la grille S est :

```
P = "123456789" °- S °:: CMAP :: (S ? ".")
```

Le tout en une seule ligne, efficace non ? d:o-)

Traitons le point -d-

Commençons par trouver sur quelle position libre il faut travailler. Tout simplement, ce sera la position possédant la plus petite liste de candidats.

si P est la liste de candidats obtenu à l'étape -c-, le nombre de candidats possibles à chaque position est simplement

```
P1 = °Longueur(P)
```

Le plus petit nombre est naturellement Min(P1). Les rangs des cases (dans la liste P) qui correspondant à ce plus petit nombre est
P1?Min(P1)
Il suffit de prendre n'importe laquelle de ces cases, elles sont équivalentes à ce stade. Disons la première :

```
x = ^(P1?Min(P1))
```

Il faut maintenant construire une suite de grilles dérivées de notre grille S, où le seul changement sera l'arrivée de chacun des candidats possibles en position x.

Ces candidats sont simplement P::x, qui est une chaîne de caractère groupant les candidats. Mettons les sous forme de liste, un caractère par liste : $(P::x).

Il faut trouver à quelle position de la grille S correspond la case désignée par x, qui est - attention à ce point - le rang de la case dans la liste L des cases libres. Son rang, dans S, est L::x. Si on se rappelle que L = S?".", alors on obtient la liste de grilles désirée par une distribution de la fonction Change sur la chaîne S (qui est unique), sur l'index de la "meilleure" case libre, qui est aussi unique, et sur la liste des candidats à cette case libre $(P::x) qui est la seule liste sur laquelle la distribution va opérer.

```
LS1 = °Change(S,(S?".")::x,$(P::x))
```

A ce stade, la liste LS1 contient une liste de grilles correspondant à tous les choix possibles pour remplir la case libre de S choisie.

Et après ?

L'étape suivante consistera à repartir de la liste LS1 qui contient désormais n grilles, et de refaire exactement la même chose pour chacune d'elles, afin d'obtenir une nouvelle liste LS2. Construisons une fonction etape qui rassemble les calculs déjà vus, et passons CMAP en variable globale _CMAP afin d'éviter de devoir passer cette gigantesque grille en argument.

```
%etape(S)
  P = "123456789" °- S °::_CMAP::(S?".")
  P1 = °Longueur(P)
  x = ^(P1?Min(P1))
%return(°Change(S,(S?".")::x,$(P::x)))
```

Il suffit d'en distribuer l'effet sur la liste LS1

LS2 = °etape(LS1)

Plus précisément, la liste LS2 contient une liste de liste de grilles correspondant à tous les choix possibles pour remplir les deux premières cases libres. Ce n'est pas tout à fait le format souhaité. Pour revenir à une simple liste de grilles, qui est le format de notre liste LS1 d'origine, utilisons une remise globale à plat.

LS2 = µ(°etape(LS1))
Notez bien que, si on adopte le même formalisme, nous avions déjà :

LS1 = µ(°etape(LS))
avec LS = {S} : liste d'un seul élément égal à S

Et l'étape suivante sera
LS3 = µ(°etape(LS2))

C'est clair, il faut faire autant d'étapes que de cases libres. L'algorithme complet est donc :

```
LS = {S}
_CMAP = (°Cmap°(%81,%81) µ 81) °? 1 //construction de la liste CMAP
Tantque(Contient(^LS,".")) // boucle tant qu'il y a des cases libres...
  LS = µ(°etape(LS)) //exécuter une étape sur la liste LS
Fintantque
```

Le sudoku est résolu en quelques lignes de programme, ne mettant en oeuvre qu'une seule boucle d'instructions .
Pour les puristes des listes, cette boucle tanque-fintantque est une laideur. Il est facile de l'éliminer avec une fonction récursive :

```
%Solution(Z)
  if (Contient(^Z,"."))  Z = Solution(µ(°etape(Z)))
  endif
%retour(Z)
```

Et le programme se résume alors à ces DEUX lignes

```
_CMAP = (°Cmap°(%81,%81)µ81)°?1
Grille = Solution({S})
```

Notez qu'aucune structure de boucle n'est utilisée ni dans le programme principal, ni dans les fonctions. Ce n'est pas seulement pour faire joli. En effet, dans des programmes qui ne sont pas convertis en code machine natif compilé par un compilateur (ce qui est le cas de EEA, mais aussi de tous les langages de script), les boucles d'exécutions tantque-fintanque etc.. consomment beaucoup de temps machine, car elles sont exécutées au niveau le plus élevé, c'est à dire dans le script écrit par l'utilisateur. Dans une programmation en listes, ces boucles existent toujours (il n'y a pas de magie), mais elles sont exécutées dans le code noyau de EEA, qui lui est en langage machine natif compilé. Donc BEAUCOUP plus rapide... La seule contrepartie est que le langage de liste utilise des données structurées potentiellement de grande taille (tout une liste), alors qu'un langage classique opère au niveau d'un seul élément à la fois. Donc un programme écrit "en listes" peut consommer beaucoup de mémoire....

performances

En passant en multi_thread la fonction etape, la grille réputée "record du monde" est résolue en environ 0,5s sur une machine assez puissante (core i7 à 3 ghz), avec une consommation mémoire à 850 Koctets tout à fait dérisoire.

CONCLUSION

Nous avons abordé dans cet exemple réel quelques caractéristiques de la programmation en liste :
- des programmes significativement courts, rapides à l'exécution, parfois gourmands en mémoire.
- une syntaxe plutôt intimidante pour les non-initiés (et encore, essayez donc l'APL...).
- des programmes élégants mais difficiles à comprendre sans une très solide documentation.
- des programmes très modulaires, robustes devant les évolutions : vous aurez noté le peu de signes modifiés entre les 3 versions programmées dans notre exemple.

Chacun verra midi à sa porte. Vous pouvez utiliser EEA de manière classique, en programmation structurée conventionnelle. Vous pouvez vous mettre aux listes.... mais attention dans ce cas, l'addiction arrive très vite !

d:o)

Code source complet du programme de résolution

```
#include(affichesudoku.eea)
//===== grille de Sudoku à résoudre, niveau de difficulté extrême (supérieur
à 11)
S = "\
1......89\
.....9..2\
......45.\
..76.....\
.3..4....\
9....2..5\
..4.7....\
5....8.1.\
.6.3....."

%Lcb(ix)
 q,m = (ix-1)/9,(ix-1)%9
%return({q,m,(m/3)+3*(q/3)})

%Cmap(i,j)
%return(||°(Lcb(i)°==Lcb(j)))

%etape(S) thread
  P = "123456789" °- S °::_CMAP::(S?".")
  P1 = °Longueur(P)
  x = ^(P1?Min(P1))
%return(°Change(S,(S?".")::x,$(P::x)))

%Solution(Z)
  if (Contient(^Z,"."))  Z = Solution(µ(°etape(Z)))
  endif
%retour(Z)

_CMAP = (°Cmap°(%81,%81)µ81)°?1
Affiche({S},"Problème")
Affiche(Solution({S}),"Solution")
```

Liste des fonctions opérant sur listes, utilisées dans ce programme

Change(L,ix,v) = liste L, modifiée au rang d'index ix, où l'on force la valeur v.
°fff(L,M) = distribution d'une fonction fff sur les éléments des listes L et M
||°(L) = réduction d'une liste par l'opérateur || (ou logique)
L °== M = distribution de l'opérateur == (par exemple) sur les éléments des listes L et M
°fff°(L,M) = produit externe d'une fonction fff sur les éléments des listes L et M
%n = liste des nombres de 1 à n
L% = nombre d'éléments d'une liste
$c = liste des caractères de la chaîne c
L$ = regroupement des éléments de la liste L en une seule chaîne de caractères
L µ n = regroupement d'une liste L par sous-liste de n éléments chacune.
µL = remise à plat d'une liste de listes imbriquées, sous forme de liste simple à un seul niveau.
L ? x = recherche d'un élément x dans une liste L
L - M = enlève de L les éléments présents dans M
L :: M = liste des éléments de L indexés par M
L :: ix = l'élément de L indexé par ix
%<L : tri numérique d'une liste

^L : 1er élément d'une liste

Fonctionnalités EEA avancées

Fonctionnalités EEA avancées

Nota : cet article s'adresse aux personnes désireuses d'exploiter EEA dans ses derniers retranchements. La lecture de ce chapitre, qui nécessite une certaine maîtrise des concepts de base de la programmation, n'est en aucun cas indispensable pour utiliser le langage EEA.

a) Fonctions

EEA permet la création de nouvelles fonctions, que vous pouvez ensuite utiliser dans vos programmes exactement de la même manière que les autres fonctions prédéfinies . Les chapitres suivants vous guideront dans la mise en œuvre de ces possibilités extrêmement puissantes.

1. Pourquoi définir une nouvelle fonction
2. La directive #include
3. Comment définir une nouvelle fonction
4. La programmation modulaire, les variables locales
5. Variables globales
6. La programmation récursive.
7. Le bloc %xxxx - %retour.

b) La programmation multi-tâches

1. Pourquoi faire du multi-tâches?
2. Comment faire du multi-tâches?
3. Les valeurs futures
4. La tentation des variables globales
5. Le gestionnaire de tâches

c) Pointeurs, tableaux, dictionnaires, listes

variables dynamiques, pointeurs.

Tableaux, listes, bases de données, dictionnaires Python

La gestion de listes

d) Mise au point des programmes EEA

EEA dispose de certains outils d'aide à la mise au point et au débogage.

Mise au point des programmes

Pourquoi définir une fonction ?

Pourquoi définir une définir une nouvelle fonction

La définition de fonction augmente la lisibilité et la maintenabilité des programmes, en isolant dans un bloc de code bien identifié une fonctionnalité dont vous avez besoin.

Elle augmente aussi la compacité de votre code si cette fonctionnalité est utilisée à plusieurs endroits de votre programme. Vous définissez une fois, vous utilisez autant de fois que nécessaire.

Elle augmente enfin la fiabilité et la vitesse de programmation, car vous pouvez réutiliser des fonctions déjà codées et éprouvées dans d' autres programmes, simplement en réutilisant le bloc de définition de la fonction.

Un bon exemple de fonction réutilisable, vous est proposé avec le fonction Nel, qui signifie "Nombre En Lettres", et qui permet la traduction en lettres de tout nombre inférieur à 2 milliards et des poussières, et dont le source en langage EEA est livré avec AJL.

Par exemple Nel(1954142857) = "un milliard neuf cent cinquante quatre millions cent quarante deux mille huit cent cinquante sept"

La fonction Nel

La directive #include

La directive #include

Nous avons vu que l'emploi de fonctions permet de réutiliser des fonctions déjà codées en réutilisant le bloc de définition de la fonction.
Pour celà, inutile de jouer du copier - coller.

Utilisez plutôt la directive `#include`. Elle agit comme un aspirateur. Codez `#include`(nomdefichier), par exemple en début de programme, afin d'aspirer toutes les lignes du fichier nomdefichier, qui viennent remplacer la directive `#include`. Plusieurs directives `#include` peuvent être utilisées.

Si vous ne précisez pas le chemin d'accès complet au fichier, il sera recherché par défaut dans le même répertoire que celui où se trouve le programme AJL, c'est à dire le répertoire où se trouve l'exécutable AJL.exe.

Attention, #include est traité par le pré-processeur EEA qui n'est donc pas soumis aux contraintes du traitement des caractères d'échappement. Les noms de fichiers sont écrits simplement comme dans l'exemple ci-dessous, sans redoublement du '\'.

Exemples

```
#include(Nel.eea)
#include(C:\AJL\scripts\Nel.eea)
```

Vous pouvez utiliser des `#include` imbriqués, c'est à dire qu'il est possible d'utiliser à nouveau une ou plusieurs directives `#include` dans un fichier déjà aspiré par une autre directive `#include`.

Comment définir une fonction

Comment définir une nouvelle fonction

Pour définir une nouvelle fonction, commencez par lui choisir un nom. Par exemple «`MaFonction`». Vous devez choisir un nom qui n'est pas déjà utilisé dans le langage EEA comme mot-clé (`liredico` par exemple) ou fonction prédéfinie (`Longueur` par exemple). Les noms de fonctions sont sensibles aux différences minuscules – majuscules.

Vous devez ensuite choisir la suite et le nom des arguments que la fonction va utiliser comme données d'entrée, puis la valeur rendue (ou la suite de valeurs) par la fonction.
Ces choix vous permettent de coder l'instruction d'entrée dans votre fonction :

```
%MaFonction(suite d'arguments séparés par des virgules)
```

Et l'instruction de sortie :

```
%retour(suite des expressions de retour)
```

Vous pouvez utiliser `%return` en synonyme de `%retour`, par analogie avec l'instruction éponyme du langage « C ».
La suite d'arguments d'entrée doit être une suite de variables, ou une suite d'éléments de tableau :

```
%Mafonction(arg2,Tab2[1],Tab2[arg1],arg1)
```

La suite des expressions de retour peut être une suite d'expressions aussi complexe que vous le souhaitez :

```
%retour(x,2*(y+Longueur(mot)))
```

Entre ces deux balises, il vous suffit d'écrire les instructions nécessaires à l'exécution de votre fonction. Aucun effet n'est produit par la simple déclaration d'une fonction, car le code présent entre les balises n'est exécuté que si la fonction déclarée est appelée quelque part dans le programme. Il est cependant nécessaire de définir les fonctions avant leur premier usage, il est donc conseillé de les regrouper à la suite les unes des autres en tout début de programme.

Exemple : On souhaite écrire une fonction qui compte le nombre d'occurrences d'une certaine lettre dans un mot. Le nom de la fonction sera « `CompteLettre` ». les deux arguments nécessaires en entrée seront le `mot`, et la `lettre` dont il faut compter le `nombre` d'apparitions dans le `mot`. La valeur de retour sera ce `nombre`.
Les balises d'entrée et de sortie seront donc :

```
%CompteLettre(mot,lettre)
%return(nombre)
```

Entre les deux balises, il vous suffit d'insérer les instructions nécessaires. Une possibilité parmi bien d'autres est :

```
%CompteLettre(mot,lettre)
nombre = 0 //initialiser le compteur à zéro
  for(i = 1  ; i <=  Longueur(mot)  ; i += 1)
  //l'indice i explore le mot lettre à lettre
    si (mot::i == lettre) //ième lettre = lettre cherchée  ?
      nombre += 1 //oui  : incrémenter le compteur
    finsi
  endfor
%return(nombre)
```

L'utilisation de cette fonction sera très simple : par exemple
`x =   CompteLettre(«AIDE AUX JEUX DE LETTRES  »,«E»)`
donnera `x = 5`

Vous pouvez définir des fonctions qui rendent plusieurs valeurs. Par exemple

```
%Decoupe(phrase)
%return(Debut(phrase,10), Fin(phrase,1))
```

Cette fonction renvoie les 10 premières lettres de la variable `phrase` d'entrée, et sa dernière lettre.

Pour utiliser un tel résultat multiple dans une instruction valide, il faut utiliser une affectation multiple pour deux variables :

```
DebutMot , DerniereLettre  =  Decoupe(«AIDE AUX JEUX»)
```

On peut aussi utiliser une fonction qui « consomme » deux arguments d'entrée. Comme notre fameuse fonction CompteLettre. Ainsi on peut faire directement :

```
X = CompteLettre(Decoupe(«AIDE AUX JEUX»))
```
Qui va donner X = 1 (la dernière lettre X est présente une seule fois dans les 10 premiers caractères de `«AIDE AUX J»`

Toute fonction peut être utilisée ainsi. Si une fonction F1 possède un argument d'entrée et 3 de sortie, et F2 trois d'entrée et un de sortie (comme la fonction standard `Partie`), il est valide de former l'expression `X = F2(F1(x))`

Programmation modulaire, variables locales

La programmation modulaire, les variables locales

Vous avez déjà vu les avantages de la programmation par fonction dans le paragraphe « Pourquoi définir une nouvelle fonction », et notamment l'usage de la directive #include.

Pour offrir ces avantages, le code de la fonction ne doit pas être adhérent au reste du code de vos

programmes. A cet effet, les variables utilisées dans une fonction sont locales à la fonction. Et de même, les variables utilisées en dehors d' une fonction lui sont absolument inaccessibles. Le seul lien d' une fonction avec l' extérieur est effectué «par valeur».

Ce terme technique « par valeur » signifie que lors de l' appel d' une fonction `Fonction(liste d'arguments)` les valeurs des arguments d' entrée servent à initialiser les variables décrites dans la balise d' entrée `%Fonction(liste des variables)` par le moyen d' une affectation multiple implicite
`liste des variables  = liste des valeurs des arguments passés`

De même, les valeurs de sortie `%return(liste d'arguments)` sont transmises «par valeur» à la fonction appelante.

Ces mécanismes assurent une totale étanchéité entre les variables locales d' une fonction et le reste du programme. Notez également que les variables locales définies et valorisées dans une fonction ne sont pas persistantes. D'une exécution à l'autre de la même fonction, aucune trace des "anciennes" variables locales n'est conservée. (pour les programmeurs aguerris, cela veut dire que EEA ne génère pas de variables statiques)

Exemple :

```
%fonction(x,y) //voir commentaire 1)
z = x + y      //voir commentaire 4)
%Return(z+1)   //voir commentaire 5)

z = 0 //voir commentaire 2)
a = fonction(z,1) //voir commentaires 3,4  5,6
a = fonction(z,1) //voir commentaire 7)
z = z + 1         //voir commentaire 8)
```

Déroulement de ce programme :

1) La fonction %fonction est ajoutée à la liste des fonctions EEA.
2) La variable z est créée dans le programme principal, et sa valeur est fixée à 0
3) La fonction « fonction » est appelée, avec deux arguments de valeur 0 et 1
4) Dans la fonction, une variable locale z est créée avec la valeur 0 + 1 = 1. Cette variable z n' a aucun rapport avec le variable de même nom dans le programme principal.
5) Une valeur retour 1 +1 = 2 est renvoyée.
6) Une variable a est créée dans le programme principale, et sa valeur est 2
7) Les étapes 3 4 5 6 sont répétées. <u>La valeur de z reste inchangée, à 0</u>.
8) La valeur 0+1 = 1 est affectée à la variable z.

Variables globales

Les variables globales

Le cloisonnement strictement local du domaine de validité des variables crée certains inconvénients. Quand certaines valeurs sont utiles dans plusieurs fonctions, la seule recette pour les partager consiste à les ajouter à la liste des arguments d'entrée et de sortie de ces fonctions. Il est aussi fréquent d'avoir besoin de définir de simples constantes utilisées dans plusieurs fonctions. Il est alors nécessaire de réécrire les affectations de définition de la valeur de ces constantes dans chaque contexte où elles sont utiles, ce qui est en flagrante contradiction avec les objectifs des fonctions (maintenance facilitée, réutilisation). Sans parler de l'aspect performance.

Pour déserrer cette contrainte, EEA propose un mécanisme de variables globales. Une variable globale traverse tous les cloisonnements et peut être partagée librement en lecture et écriture partout dans n'importe quel contexte : fonctions , programme principal.

EEA identifie simplement une variable globale par une norme de nommage : **le nom d'une variable globale commence par un '_'** ("souligné", dit aussi "tiret du 8"). Un tableau ou une liste peuvent aussi être définis comme globaux, selon la même norme.

Enfin, EEA peut fonctionner dans un mode où toutes les variables sont automatiquement considérées comme globales, indépendamment de leur nom. Il suffit d'inclure dans le programme la directive suivante :

```
#global
```

Attention à utiliser cette directive à bon escient : elle interdit absolument l'usage des fonctions récursives et des threads qui nécessitent impérativement un cloisonnement local des variables. Utiliser des globales dans des threads ne vous conduira qu'à des résultats inconsistants, imprévisibles, nécessitera de vains efforts de mise au point, et au final apportera désillusions et frustrations. Vous ne direz pas que vous n'étiez pas clairement prévenus !

Programmation récursive

La programmation récursive.

EEA permet la programmation récursive, c'est-à-dire qu' une fonction peut s' appeler elle-même. C' est une technique de programmation extrêmement puissante mais un peu délicate à manier. Abordons là par un exemple :

En mathématique, la fonction factorielle d' un nombre est le produit de ce nombre par tous les entiers compris entre 1 et ce nombre. Par exemple :

```
factorielle(1) = 1
factorielle(2) = 2*1 = 2
factorielle(3) = 3*2*1 = 6
factorielle(4) = 4*3*2*1 = 24
factorielle(5) = 5*4*3*2*1 = 120
```

etc…

On constate que

```
factorielle(5) = 5*factorielle(4)
factorielle(4) = 4*factorielle(3)
factorielle(3) = 3*factorielle(2)
factorielle(2) = 2*factorielle(1)
```

`factorielle(1) = 1`

Un peu de réflexion conduit à s'assurer que c'est toujours vrai :
`factorielle(n)= n*factorielle(n-1),`
à la seule condition que n soit supérieur à 1. En effet,
`factorielle(1) = 1*factorielle(0)`
conduirait à une difficulté sur la valeur de factorielle(0).

On voit donc qu'on peut définir la valeur d'une factorielle en s'appuyant

1) sur la valeur de la factorielle du nombre immédiatement plus petit : `factorielle(n) = n * factorielle(n-1),`
2) et sur la valeur d'un seul cas particulier : `factorielle(1) = 1`

Ces deux caractéristiques sont celles d'une définition récursive, que l'on peut directement utiliser dans la définition de fonction suivante :

```
%factorielle(n)
si(n<= 1)
  f = 1
sinon
  f = n*factorielle(n-1)
finsi
%return(f)
```

On voit que le code de la fonction factorielle fait usage (auto-référence) de sa propre fonction. C'est cela une fonction récursive.

Vous êtes allergiques aux maths et préférez un exemple dans le domaine des lettres ?
Qu'à cela ne tienne, pour l'amusement, définissons notre fonction CompteLettre de façon récursive.

```
%CompteLettre(mot,lettre)
nombre = (mot::1 == lettre)
si (Longueur(mot) <= 1)
  exit
sinon
  nombre += CompteLettre(mot<<1,lettre)
finsi
%return(nombre)
```

Cette fonction fait appel à plusieurs astuces de programmation. Explications :

`nombre = (mot::1 == lettre)`
on compare la première lettre de mot `(mot::1)` à `lettre`. Si le résultat est vrai, l'opérateur de comparaison == donne la valeur booléenne VRAI qui est en réalité le nombre 1. S'il est faux, la valeur FAUX = le nombre zéro. On a donc traité le comptage demandé pour la seule première lettre du mot.

```
si (Longueur(mot) <= 1)
  exit
```
Si le mot ne fait qu'une lettre ou moins, on a donc le résultat demandé. On termine immédiatement la fonction par l'ordre exit, qui force l'exécution sans délai de la fonction de sortie `%return`.
Notez l'analogie avec l'exemple factorielle plus haut : On traite ici de manière explicite un cas particulier très simple.

`nombre += CompteLettre(mot<<1,lettre)`

Tout se passe ici. Si le mot fait plus d' une lettre, on indique simplement que le compte de lettres est égal au compte déjà obtenu pour la première lettre + le compte du reste du mot, calculé par récursion. La récursion permet donc de diminuer la taille du mot à traiter, ce qui va finir par le ramener à la taille du cas particulier de taille 1 que l' on sait traiter directement.

Le code `mot<<1` est une manière d' enlever la 1ère lettre d' un mot, car l'**opérateur <<** va éliminer de `mot` la lettre la plus à gauche.

Toute l' astuce de la programmation récursive consiste donc à :

1) identifier et traiter explicitement un cas simple .
2) identifier comment simplifier le problème lors de l' appel récursif, de sorte que la chaîne des appels récursifs finisse toujours par tomber sur le cas simple précédent.
3) Faire le lien entre le résultat cherché et le résultat obtenu par l' appel récursif (par exemple une addition pour CompteLettre, et une multiplication pour factorielle)

Des exemples plus opérationnels de programmation récursive sont donnés par la fonction Nel, et le programme Découpe mots que vous pouvez utilement analyser.

La programmation multi-tâche

La programmation multi-tâche

Pourquoi faire du multi-tâches ?

Parce que c'est le seul moyen d'utiliser la puissance des ordinateurs modernes.

En effet, pendant de longues années, augmenter les performances d'un programme, du point de vue du programmeur, n'a été qu'une question de ... patience. il suffisait d'attendre que l'utilisateur du programme change sa machine par une autre plus puissante, et le programme, même ancien, tournait plus vite. Sur une période de 20 ans, les processeurs des ordinateurs sont passés de fréquences d'horloge de quelques Mhz à quelques Ghz, soit mille fois plus vite. Le pire qui pouvait arriver, toujours du point de vue du programmeur, était la nécessité de recompiler de temps en temps son programme afin qu'il suive les évolutions du système sur lequel il s'appuyait (Windows 95, XP, Vista, 7, 8.1, etc...)

Ce modèle a été vérifié de 1973 à 2004 environ, avec un doublement des fréquences d'horloge tous les 18 mois. Mais la fréquence des processeurs tend désormais à stagner. Les constructeurs jouent depuis sur un autre tableau : le nombre de cœurs de leurs processeurs. Dès 2010, on dispose de processeurs grand-public à 6 coeurs (Intel i7 980 XE). Hélas, du point de vue du programmeur, il ne suffit plus de recompiler pour tirer parti de cette nouvelle source de puissance. Pour qu'un programme utilise en même temps plusieurs cœurs, il faut revoir en profondeur sa logique.

Par exemple, comment accélérer une recherche portant sur tout un dictionnaire, si l'on dispose de 2 cœurs ? Il faut lancer 2 tâches simultanées, l'une explorant le dictionnaire de A à M par exemple, et l'autre de N à Z. Puis attendre les deux tâches, récupérer les deux résultats partiels, et les fusionner. La recherche d'anagramme dans AJL utilise cette technique, avec un découpage du dictionnaire en environ 50 sections, chacune affectée à une tâche distincte. Un gestionnaire interne s'assure que le nombre de tâches simultanées n'excède jamais le nombre de cœurs du processeur, et une tâche de collecte a pour rôle de capter et fusionner les résultats partiels. Ces actions nécessitent la mise en œuvre de primitives systèmes complexes : lancement de threads, sémaphore de synchronisation, mutex, gestion d'événements.

Comment faire du multi-tâche en EEA

Avec EEA, tout ce côté complexe est masqué. Prenons un exemple simpliste mais illustratif. On désire effectuer un calcul sur tous les nombres de 1 à 1000. ce calcul est réalisé par une fonction utilisateur calcul(n) dont le détail importe peu.

1) version mono tâche

```
//programme principal
boucle(i=1; i<=1000; i+=1)
  calcul(i)
finboucle
```

2) version encore mono tâche, mais en découpant le travail en deux parties indépendantes. C'est l'étape de modification de la logique du programme.

```
//création d'une nouvelle fonction, pour réaliser le calcul sur les nombres de
x à y
%faireboucle(x,y)
boucle(i=x; i<=y; i+=1)
  calcul(i)
finboucle
%retour()

//nouveau programme principal
faireboucle(1,500)
faireboucle(501,1000)
```

3)idem, en 2 tâches simultanées (modification purement technique du programme)

c'est magique : il suffit d'ajouter un mot clé "thread" à la définition de la fonction faireboucle

```
%faireboucle(x,y) thread
boucle(i=x; i<=y; i+=1)
  calcul(i)
finboucle
%retour()

//programme principal inchangé
faireboucle(1,500)
faireboucle(501,1000)
```

Que va-t-il se passer ? le programme principal va donc exécuter les deux instructions

```
faireboucle(1,500)
faireboucle(501,1000)
```

Chacune de ces instructions va se dérouler dans une nouvelle tâche, indépendante de celle du programme principal. A cet instant, on aura donc 3 tâches simultanées : une pour le 1er faireboucle, une pour le second, une pour le programme principal.
Le programme principal n'attend pas les deux tâches qu'il a lancées. C'est le point fondamental qu'il faut comprendre. Ce programme principal se termine donc instantanément après avoir lancé les deux faireboucle.
Puis chacune des 2 faireboucle vit sa vie, et finira par se terminer dans un ordre qui n'est pas prédictible. Ce n'est que lorsque la dernière des tâches sera achevée que AJL considèrera que le script EEA est terminé, et « rendra la main » à l'utilisateur.

Les valeurs futures

Tout celà est bel et beau, mais comment faire si la chaque tâche doit rendre un résultat utilisé dans le

programme principal. Par exemple, imaginons que notre fonction calcul rende une valeur, et que le but du programme principal soit de sommer toutes les valeurs rendues par la fonction calcul. Reprenons notre exemple depuis le début :

1) calcul simple, version mono tâche

```
somme = 0
boucle(i=1; i<=1000; i+=1)
  somme += calcul(i)
finboucle
```

2) calcul simple, encore mono tâche, mais en découpant le travail en deux parties (modification de la logique du programme)

```
//création d'une nouvelle fonction, pour réaliser le calcul sur les nombres de
x à y
%faireboucle(x,y)
s = 0
boucle(i=x; i<=y; i+=1)
  s += calcul(i)
finboucle
%retour(s)

somme1 = faireboucle(1,500)
somme2 = faireboucle(501,1000)
somme = somme1 + somme2
```

3)idem, en 2 tâches simultanées (modification purement technique du programme)

magique, là aussi, il suffit d'ajouter un mot clé thread

```
%faireboucle(x,y) thread
s = 0
boucle(i=x; i<=y; i+=1)
  s += calcul(i)
finboucle
%retour(s)

somme1 = faireboucle(1,500)
somme2 = faireboucle(501,1000)
somme = somme1 + somme2
```

Là, normalement, vous devez vous frotter les yeux et crier à l'escroquerie.

En effet : "somme1 = faireboucle(1,500)" lance une tâche "faireboucle", fort bien. Mais le programme principal est supposé se continuer sans attendre cette 1ère tâche, sinon ca ne servirait pas à grand chose de faire du multi tâche! Et dans ce cas, quelle valeur affecter à la variable somme1, puisqu'on ne connaît pas encore le résultat de la tâche faireboucle !?!?

C'est là que EEA fait intervenir la notion de **valeur future**. Le résultat de la 1ère tâche faireboucle est une valeur future, non connue, dont la tache faireboucle fixera plus tard la valeur réelle. Une valeur future est associée à une tâche exécutée en parallèle du programme où elle est utilisée. Rien n'interdit de transférer cette association à une variable, en l'occurrence la variable somme1, qui devient dès lors une variable future, associée au résultat de la 1ère tâche faireboucle.

le programme EEA se poursuit alors avec l'instruction suivante : "somme2 = faireboucle(501,1000)"
De même, somme2 devient une variable future associée au résultat futur de la 2ème tâche faireboucle. Le programme EEA se poursuit avec "somme = somme1 + somme2".

Là, impossible d'avancer sans connaître les valeurs réelles de somme1 et somme2. EEA va engager, automatiquement, une boucle d'attente sur l'obtention des valeurs réelles de somme1 et somme2, c'est à dire une boucle d'attente des 2 tâches parallèles faireboucle. Dès qu'elles seront terminées, les valeurs réelles de somme1 et somme2 deviendront disponibles, et l'instruction somme = somme1 + somme2 sera exécutée. Plus précisément, EEA autorise la poursuite du programme tant qu'une valeur future n'est utilisée que pour être attribuée à une variable (utilisation en Lvalue ou VALG). En revanche, la nécessité de connaître la valeur réelle d'une valeur future (Rvalue ou VALD), par exemple pour affichage ou pour participer à un un calcul, entraîne automatiquement une boucle d'attente.

L'intérêt des valeurs futures réside dans leur double rôle : une réservation de place qui permet au programme principal d'avancer sans attendre, et un dispositif de synchronisation automatique qui introduit une boucle d'attente quand la valeur réelle est vraiment nécessaire.

Il n'est même pas nécessaire d'attribuer chaque valeur future à une variable pour bénéficier de ce mécanisme. Le code suivant, plus direct, produit le même résultat :

```
%faireboucle(x,y) thread
s = 0
boucle(i=x; i<=y; i+=1)
  s += calcul(i)
finboucle
%retour(s)

somme = faireboucle(1,500) + faireboucle(501,1000)
```

La tentation des variables globales

Il est tentant de remplacer toutes ces histoires de valeurs futures par une simple variable globale. Reprenons notre 1er exemple.

1) pour rappel, la version d'origine, mono-tâche :

```
//programme principal
somme = 0
boucle(i=1; i<=1000; i+=1)
  somme += calcul(i)
finboucle
Message(somme)
```

2) proposition avec une variable globale _somme

```
%faireboucle(x,y) thread
boucle(i=x; i<=y; i+=1)
  _somme += calcul(i)
finboucle
%retour()

//programme principal
_somme = 0
faireboucle(1,500)
faireboucle(501,1000)
Message(_somme)
```

Rien n'interdit cette version. Cependant, ce programme ne vaut absolument rien dans un monde de tâches parallélisées. En effet, les trois instructions `_somme = 0 faireboucle(1,500) faireboucle(501,1000)` vont se dérouler en une fraction de seconde, car `faireboucle` ne provoque aucune attente dans le programme principal. Celui ci va directement arriver sur la dernière instruction `Message(_somme)` qui va capter la valeur instantanée du compteur `_somme`, qui n'aura sans doute qu'à

peine été modifié par les deux fonctions parallèles `faireboucle`.

Retenez : Utiliser des globales dans des threads ne vous conduira qu'à des résultats inconsistants, imprévisibles, nécessitera de vains efforts de mise au point, et au final apportera désillusions et frustrations. Vous ne direz pas que vous n'étiez pas clairement prévenus !

La fonction Wait()

Cette fonction vous permet d'attendre explicitement la fin d'un ou plusieurs threads, en utilisant les variables futures comme élément de synchronisation seulement. En effet, Wait ne s'intéresse absolument pas à la valeur des variables futures, mais seulement à leur disponibilité.

Exemple

```
%faitquelquechose(message) thread
  Selection("on a fait " + message)
%return(0)       //la valeur rendue, ici 0, n'a aucune importance


v1 = faitquelquechose("ceci")
v2 = faitquelquechose("cela")

Wait(v1,v2)

Selection("nous venons de terminer de faire ceci et cela")
```

Le gestionnaire de tâches

Beaucoup de programmes EEA sont des boucles, parfois imbriquées, d'exploration de dictionnaire. Par exemple le programme suivant cherche les couples de mots dont la juxtaposition est aussi un mot, comme AVANTAGE + USE = AVANTAGEUSE. (remarque : il s'agit d'un exemple didactique, certainement pas du meilleur algorithme pour obtenir le résultat cherché.)

```
liredico(mot1)
liredico(mot2)
si (Mot(mot1+mot2))
  Selection(mot1 + "+" + mot2)
finsi
finliredico
finliredico
```

Une approche multi tâche brutale mais parfaitement efficace consiste à désirer une tâche séparée pour chaque mot1 de la boucle liredico de 1er niveau :

```
%boucledico(mot1) thread
liredico(mot2)
si (Mot(mot1+mot2))
  Selection(mot1 + "+" + mot2)
finsi
finliredico
%retour()

liredico(mot1)
  boucledico(mot1)
finliredico
```

Le gestionnaire de tâche intégré à EEA va faire en sorte de ne pas lancer plus de tâches que n'en peut absorber votre ordinateur. Aucune précaution particulière n'est à prendre par le programmeur EEA sur le nombre de tâches créées. En pratique, je recommande cette méthode : écrivez une boucle de lecture du dictionnaire dans votre programme principal, et pour chaque mot, faites appel à une fonction "thread". C'est la méthode la plus simple à mettre en oeuvre, et elle est très efficace.

Il en est de même pour les valeurs futures. Si à la place d'un affichage par Selection(...) on souhaite savoir combien de mots juxtaposés on peut trouver, il suffit d'écrire

```
%boucledico(mot1) thread
s = 0
liredico(mot2)
si (Mot(mot1+mot2))
  s += 1
finsi
finliredico
%retour(s)

liredico(mot1)
  Nbre[mot1] = boucledico(mot1)
finliredico

Total = 0
liredico(mot1)
  Total += Nbre[mot1]
finliredico
```

La 1ère boucle liredico(mot1) va lancer les tâches parallèles nécessaires, chacune associée à une variable future Nbre[mot]. Le 2ème boucle liredico attend chaque valeur future et en effectue la somme.

ATTENTION ! ATTENTION !
Notez bien qu'il ne fallait surtout pas écrire "naïvement" :

```
Total = 0
liredico(mot1)
  Total += boucledico(mot1)
finliredico
```

car Total += boucledico(mot1) exige, pour être exécutée, la valeur réelle rendue par la fonction, et donc une boucle d'attente serait automatiquement exécutée par EEA après chaque appel de la fonction boucledico, ce qui ruine totalement le parallélisme espéré.

multi-tâche et programmation en listes ?

...sont faites pour s'entendre. Ainsi, si l'on souhaite exécuter une fonction `fonctionmot` sur une liste de mots, chaque mot disposant de son propre thread, il suffira d'écrire très simplement comme dans l'exemple suivant :

```
%fonctionmot(mot) thread
  ...des lignes de code ...
%retour(s)

Dicoliste(D)  //fabrique une liste avec les mots du dictionnaire en cours
LISTE = °fonctionmot(D) //calcule la LISTE résultat en multi-tâches
```

multi-tâche et récursivité ?

...sont rarement compatibles. Bien que EEA accepte parfaitement qu'un programme appelle une fonction qui appelle une tâche qui appelle une fonction, etc... il n'est pas recommandé de déclarer des fonctions récursives en multi-tâches. En effet, si nous reprenons l'exemple de factorielle (fact(n) = n * fact(n-1)), le calcul de fact(4) par exemple nécessite le fonctionnement simultané de 4 tâches : fact(4) qui appelle facf(3), qui appelle fact(2) qui appelle fact(1), chacune attend la suivante car la valeur future fact(n-1) est nécessaire pour le calcul de n * fact(n-1) et donc la fin de tâche de fact(n). De sorte que si le nombre de tâches nécessaires excède le nombre de tâches total admissible sur votre ordinateur, le calcul ne pourra jamais s'effectuer, et EEA entrera dans une boucle d'attente infinie...

conclusion

EEA offre un mécanisme rudimentaire mais efficace de programmation parallèle. Sa mise en œuvre se résume à l'identification du bloc de code qui s'exécutera dans une tâche indépendante, sous la forme d'une fonction dotée du mot clé "thread", et des éventuelles valeurs futures associées. Il n'en demeure pas moins que le programmeur reste responsable de l'adaptation de la logique de son programme à un contexte multi-tâche.

variables dynamiques, pointeurs

Variables dynamiques, pointeurs

EEA permet la désignation indirecte de variables par deux méthodes. La première méthode consiste simplement à utiliser une "variable dynamique" contenant le nom d'une autre variable. La seconde méthode correspond à l'utilisation d'un "pointeur de variable". Absolument rien ne prédestine une variable à servir de de nom de variable ou de pointeur de variable, rappelez vous, EEA est un langage complètement non typé qui n'exige aucune pré définition des types de données.

1) Les variables dynamiques

Dans l'exemple ci dessous, la variable p servira de variable dynamique désignant z.

```
 z = 1          //une variable z est créée, et sa valeur fixée à 1
 p = "z"        //une variable p est créée, elle contient le nom de
la variable z
```

Afin d'utiliser le contenu d'une variable comme nom d'une autre variable, il suffit d'utiliser l'opérateur "pointeur" noté *****

```
 p = "z"        //une variable p est créée, elle contient le nom de
la variable z
 *p = 2         //la variable désignée par p est assignée à la valeur
2
```

La trace d'exécution de ces deux lignes est :

```
[Trace] p = "z"
[Trace] z = 2
```

De même, on peut utiliser un pointeur en partie droite (VALD). par exemple

```
 p = "z"        //une variable p est créée, elle contient le nom de
la variable z
 *p = 2         //la variable désignée par p est assignée à la valeur
2
```

```
 x = *p          //la variable x est assignée à la valeur désignée par
la variable p
 z = 3           //la variable z est assignée à la valeur 3
 x = *p          //la variable x est assignée à la valeur désignée par
la variable p
```

La trace d'exécution de ces cinq lignes est :

```
[Trace] p = "z"
[Trace] z = 2
[Trace] x = 2
[Trace] z = 3
[Trace] x = 3
```

Autre exemple, mettant en œuvre une notion de vecteur

```
 vecteur = "[1][2]"
 base = "Tableau"
 *(base+vecteur) = 2
```

La trace d'exécution de ces trois lignes est :

```
[Trace] vecteur = "[1][2]"
[Trace] base = "Tableau"
[Trace] Tableau[1][2] = 2
```

La désignation d'une variable par son nom est résolue dans l'espace de variables (global ou local) dans lequel on se situe. Ainsi, un nom local "z" est toujours local s'il est utilisé à l'intérieur d'une fonction, quand bien même l'affectation p = "z" aurait elle été effectuée en dehors de cette fonction. Dit autrement, le mécanisme de variable dynamique n'apporte aucune modification au fonctionnement du cloisonnement entre variables locales. Par exemple :

```
%affecte(p)
//une variable locale nommée par p est créée, puis assignée à la
valeur 2, puis détruite dans le contexte local de cette fonction
  *p = 2
%return()

z = 1
affecte("z")
Message(z)
```

ceci écrira la valeur 1, car la variable z est dans un contexte local différent de l'espace de variables utilisé par la fonction "affecte"

2) Les pointeurs de variables

Nota bene : cette possibilité est livrée à tire expérimental. (AJL 5.6.5)

Dans l'exemple ci dessous, la variable p servira de pointeur de variable, désignant z. Il utilise le nouvel **opérateur unaire &**, appelé opérateur adresse.

```
 z = 1          //z est créé, et sa valeur fixée à 1
 p = &z         //p est créée, elle contient un pointeur qui est
l'adresse de la variable z
```

On peut alors utiliser le pointeur p, de manière très similaire à celle montrée plus haut, pour lire ou modifier la valeur de la variable z, à l'aide de l'opérateur "pointeur" *

```
 *p = 2          //la variable z pointée par p est assignée à la
valeur 2
```

De même, on peut utiliser un pointeur en partie droite (VALD). par exemple

```
 p = &z         //une variable p est créée, elle contient un pointeur
sur z
 *p = 2         //z, variable pointée par p est assignée à la valeur 2
 x = *p         //la variable x est assignée à la valeur pointée par
p, soit la valeur de z
```

A la différence des variables dynamiques, un pointeur désigne toujours la variable dans l'espace de variables (global ou local) dans lequel l'affectation pointeur = &variable a été faite.

```
%affecte(p)
 *p = 2         //la variable pointée par p est assignée à la valeur 2
%return()

 z = 1
 affecte(&z)    //appel de la fonction "affecte" avec comme argument
un pointeur sur z
 Message(z)
```

ceci écrira la valeur 2, car l'adresse &z passe outre tous les cloisonnements entre espaces de variables.

Aucune opération (addition, etc..) n'est permise sur ces pointeurs. Comme dans tout langage manipulant des pointeurs, il est recommandé la plus grande prudence avec l'usage de cette puissante mais dangereuse possibilité. Dans l'état actuel du langage EEA, les seules applications utiles que j'ai notées sont :
- une alternative au décloisonnement local, permettant le partage d'une même variable par une ou plusieurs fonctions, sans pour autant la doter d'un statut global. Voir l'exemple ci-dessus.
- une astuce permettant de ne pas appliquer une distribution sur l'ensemble des éléments d'une liste.
Exemple :

Si L et M sont des listes, °f(L,M) distribue la fonction f sur chacun des éléments de L et M. Par exemple

```
%f(x,y)
%return(x + y)
L = %4
M = %4

R = °f(L,M)
```

donne R = {2,4,6,8}

Il est impossible d'appliquer la fonction sur la liste M d'une part, et sur chacun des éléments de L d'autre part. Sauf si on ruse avec un pointeur :

```
%f(x,y)
%return(x + *y)
```

```
L = %4
M = %4

R = °f(L,&M)
```

donne R = {{1,1,2,3,4},{2,1,2,3,4},{3,1,2,3,4},{4,1,2,3,4}}

Tableaux, listes, bases de données, dictionnaires

Tableaux et dictionnaires Python

EEA comporte de nombreuses similitudes (non volontaires, figurez vous) avec le langage moderne Python. Notamment l'entité "dictionnaire" de Python ressemble énormément aux tableaux de EEA. Par exemple, les quelques lignes de EEA ci dessous correspondent avec ce que l'on ferait en Python avec 3 "dictionnaires" `Tigre`, `Singe` et `Girafe`, et la création dans ces dictionnaires des clés "poids" et "taille", affectées de diverses valeurs.

```
Tigre["poids"] = 200
Tigre["taille"] = 100
Singe["poids"] = 70
Singe["taille"] = 120
Girafe["poids"] = 1000
Girafe["taille"] = 500
```

L'accès direct à une des valeurs contenue de ces "dictionnaires", sans à priori en connaître à l'avance le nom, se fait très simplement avec un pointeur, opérant sur la partie nom du tableau EEA. Par exemple, dans la continuité de 6 lignes EEA précédentes :

```
a = "Tigre"
b = *a["poids"]

[Trace] a = Tigre
[Trace] b = 200
```

voir la fonction Existe

Tableaux, Listes, Bases de données

Quand utiliser une liste, et quand utiliser un tableau avec un indice ?

C'est très simple : si votre indice est numérique, une liste sera toujours plus rapide qu'un tableau. En effet, l'implémentation EEA de **`Data::x`**, où x est un entier et Data une liste, sera toujours plus performante que **`Data[x]`**, où x est un entier est Data un tableau.

Évidemment, si x est une chaîne de caractère, vous n'avez pas le choix, il faut utiliser un tableau.

Gérer une base de donnée en mémoire

Une activité très courante en programmation consiste à organiser des données en structures, et à effectuer des recherches d'une donnée particulière au sein de la structure. EEA permet très simplement d'organiser des données selon une véritable base de données facile à interroger. Par exemple, vous souhaitez gérer un répertoire de noms.

```
EXEMPLE 1
```

- Créez une liste :

```
Repertoire = {{"Inconnu","?"}, {"Dupont","06.54.32.21.10"},
{"Martin","01.23.45.56.67"}, {"Durand","04.98.87.65.54"}}
```

c'est une liste de 4 listes. Chacune des 4 listes contient un nom (unique) et une donnée numéro de téléphone.

- Créez un index :

```
Index["Dupont"] = 2
Index["Martin"] = 3
Index["Durand"] = 4
```

Les valeurs 2,3,4 correspondent aux rangs des 3 personnes connues dans la liste `Repertoire`

Pour trouver la donnée téléphone à partir d'une variable N contenant un nom, c'est très simple et très rapide, une seule ligne suffit :

```
phone = Repertoire::Existe(Index[N],1)::2
```

Explications
si N = "Martin"

Existe(Index[N],1) est égal à Existe(Index["Martin"],1). Or Index["Martin"] existe, la valeur rendue est donc Index["Martin"] = 3
donc Repertoire::3 = {"Martin","01.23.45.56.67"} (3ème enregistrement de la liste `Repertoire`)
et {"Martin","01.23.45.56.67"}::2 = "01.23.45.56.67"

si N = "Zorglub"
Index["Zorglub"] n'existe pas, la valeur rendue est donc celle par défaut, indiquée à 1 (deuxième argument de la fonction Existe)
donc Repertoire::1 = {"Inconnu","?"}
et {"Inconnu","?"}::2 = "?"

```
EXEMPLE 2

//Organiser le dictionnaire en base de données
ix = 1
Dico = {} //liste vide
Liredico(m)
  Index[m] = ix++ //ajouter le mot m en indice du tableau
Index
  Dico += m // ajouter le mot m dans la liste Dico
Finliredico
```

```
//Rechercher le mot situé 7 rangs plus loin dans le
dictionnaire qu'un mot x
x = "DICTIONNAIRE"
ix = Existe(Index[x],0)
si (ix)
  Selection("%s + 7 = %s",x,Dico::(ix+7))
sinon
  Message("le mot %s est absent du dictionnaire",x)
finsi
```


Gestion de listes

Spécifications détaillées du fonctionnement des listes

(1) LES LISTES DANS EEA

A l'instar de nombreux langages actuels (python, ruby, perl ...) ou anciens (citons le sublime et méconnu APL), EEA gère la notion de liste d'éléments.

Qu'est ce qu'une liste ? C'est une série ordonnée d'objets. Série ordonnée signifie que chaque élément d'une liste est repéré par son numéro d'ordre dans la liste. On appelle ce numéro un index. Le 1er élément d'une liste a l'index 1. Série ordonnée d'objets signifie qu'un élément de liste peut-être n'importe quelle entité connue de EEA : une variable de n'importe que type, une chaîne de caractère, un entier, un nombre décimal, et même ... une autre liste. Nous reviendrons plus tard sur cette intéressante possibilité de gérer des listes imbriquées.

Pour créer une liste, rien de plus simple, l'opérateur { } s'en charge. De manière intuitive : {07,"mars",1936,"Georges","Perec"} est une liste de 5 éléments, comportant des nombres et des chaînes de caractères. {"Tigre","Lion","Panthère","Escargot"} est une autre liste. De manière plus approfondie, ces valeurs sont en réalité les arguments d'entrée d'un opérateur {}. Cet opérateur donne une seule valeur retour : une entité spécifique appelée liste, qui regroupe toutes les valeurs qui lui sont données en entrée. Une liste est une sorte de boîte, ou structure pour utiliser le terme informatique.

Il existe une 2ème manière de créer une liste, bien adaptée au cœur de cible de EEA : le traitement des chaînes de caractères. L'opérateur $ appliqué à une chaîne de caractères, crée une liste comportant chacun des caractères de la chaîne:

$"azerty" --> {"a","z","e","r","t","y"}.

Enfin, adapté au traitement des nombres ou des indices, et donc des boucles d'itération, l'opérateur %n génère la liste des nombres entiers de 1 à n.

%5 --> {1,2,3,4,5}

Nota : par clarté dans ce texte, il est adopté cette convention de nommage : les noms de listes sont notés tout en majuscules. Les autres entités (nombre, chaînes, ...) tout en minuscules. Cette convention est recommandée, mais nullement obligatoire.

(2) OPERATIONS SUR LES LISTES

Un opérateur ou une fonction appliquée sur une liste peut s'appliquer à la liste en tant que contenant,
Exemple : {1,2} "ajouté à" {3,4} --> {1,2,3,4} // on additionne les listes

ou bien s'appliquer à chaque élément de la liste.
Exemple : {1,2} "ajouté à" {3,4} --> {4,6} // on additionne entre eux les éléments des listes

La règle, et les notations choisies dans EEA sont simples et homogènes :
- Les opérateurs "classiques" (+, - , =, etc..), ou les fonctions, s'appliquent toujours à la liste en tant que contenant. L'idée est de coller au plus près le mode de fonctionnement relatif aux chaînes de caractères, dont on peut considérer qu'elles sont comme des listes de simples caractères. Ainsi {"a","b"} + {"c","d"} = {"a","b","c","d"} tout comme "ab"+"cd"="abcd".
- Pour qu'un de ces opérateurs ou fonctions s'applique à chaque élément de la liste, il faut utiliser la **notation °** accolée à l'opérateur voulu. Exemple : {1,2} °+ {3,4} --> {4,6} ou {"a","b"} °+ {"c","d"} --> {"ac","bd"}
- Et nous verrons l'arrivée de quelques nouvelles instructions, spécifiques des listes.

Commençons passer en revue le fonctionnement des opérateurs qui travaillent sur les listes en tant que contenant. Nous reviendrons plus tard sur la très puissante **notation °** (rond).

(2.1) OPERATIONS SUR LES LISTES EN TANT QU'ENSEMBLES D'ELEMENTS

Dans cette catégorie, la valeur des éléments d'une liste n'est pas modifiée, mais leur nombre ou leur position dans la liste résultat peut changer. Globalement, le fonctionnement des opérateurs exposés ici est consistant avec leur fonctionnement classique sur les chaînes de caractères,qui sont déjà des entités proches de la notion intuitive de liste.

0) affectation =

Trivialement, L'opérateur = permet d'affecter une liste à une variable
LISTE1 = {1,2,3,4,5}
LISTE2 = LISTE1

1) l'opérateur +, concaténation.

Comme pour les chaînes de caractères, + est une simple concaténation.
trivialement {1,2} + {3,4,5} --> {1,2,3,4,5}

2) l'élimination -

De manière consistante avec l'opérateur - portant sur les chaînes de caractères (rappel : "ELIMINATION" - "NI" --> "ELMATO"), l'opérateur - portant sur des listes élimine du 1er argument tout élément présent dans le second argument.
{0,5,"N",3,"X",5,"NX"} - { 5,"X" } --> { 0,"N",3,"NX"}

3) l'inversion -

De manière consistante avec l'opérateur - unaire portant sur une chaîne de caractères, l'opérateur - portant sur une seule liste inverse l'ordre des éléments de la liste.
- {0,5,7,"N",3,"X",5,"NX"} --> { "NX",5,"X",3,"N",7,5,0}

4) les décalages << et >>

On éjecte les éléments d'une liste par la droite ou par la gauche. Les éléments éjectés sont simplement éliminés.
LA = {1,2,3,4,5}
LA >> 2 --> {1,2,3} (on a éjecté 2 éléments par la droite)
LA << 3 --> {4,5} (on a éjecté 3 éléments par la gauche)

5) la rotation <>

On fait cycler des éléments d'une liste par la droite ou par la gauche, les éléments sortis d'un côté réapparaissent de l'autre. Les valeurs positives font cycler de gauche à droite, les valeurs négatives de droite

à gauche.
LA = {1,2,3,4,5}
LA <> 2 --> {4,5,1,2,3} (rotation de 2 éléments sans le sens gauche->droite)
LA <> (-1) --> {2,3,4,5,1} (rotation d'un élément dans le sens droite->gauche)

6) les tris alphabétiques $> et $<, et les tris numériques %> et %<

$> trie une liste dans l'ordre alphabétique descendant (de sorte que chaque élément de la liste soit > suivant)
$>{"e","b","z","j"} --> {"z","j","e","b"}
idem pour le tri ascendant $<

%< trie une liste dans l'ordre numérique ascendant (de sorte que chaque élément de la liste soit < suivant)
%<{5,1,5,3,4} --> {1,3,4,5,5}
idem pour le tri descendant %>

7) la sélection ::

Sélectionne les éléments d'une liste en fonction d'une liste d'index, ou d'un index unique. Les deux opérandes de l'opérateur :: peuvent ainsi être, selon les besoins, des listes ou des éléments "classiques".Exemples :
LA = {11,22,33,44,55}
LA :: 3 --> 33 //l'opérande droit est une élément simple --> le résultat est un élément simple
LA :: {3} --> {33} //l'opérande droit est une liste --> le résultat est une liste
LA::{2,4,2} --> {22,44,22} //idem

Avec un opérande gauche de type chaîne de caractères et un opérande droit de type liste, on obtient un nouvel opérateur intéressant travaillant sur les chaînes de caractères.
ch = "azerty"
ch::{2,1,2,1} --> "zaza"

8) le masque &

Sélectionne les éléments d'une liste correspondants aux 1 de même rang dans l'autre liste . Exemple :
M = {0,1,1}
L = {"AF","DEF",DG","XZ"}
L & M = {"DEF","DG"}
les éléments 2 et 3 de L ont été sélectionnés car les 1 cherchés sont en position 2 et 3 dans M.
Usuellement, l'opérande M s'obtient par l'action d'opérateurs booléens sur des listes. Cette fonction masque, couplée à une fonction de sélection booléenne, devient un très puissant outil de sélection d'éléments dans une liste.

9) correspondance entre deux listes, opérateur ?

L ? X
Argument L à gauche : liste à explorer, à la recherche d'éléments. Argument X à droite : liste des éléments recherchés.
Pour chaque élément de X, on obtient un index i si cet élément est égal au ième élément de L. La taille du résultat est la même que l'argument droite X.
L = {15,40,63,18,27,40,33,29,40,88}
X = {29,63,40,33,50}
alors L ? X --> {8,3,2,7,11}
notez la valeur 11 qui signifie que 50 est non trouvé dans la liste des 10 éléments de L. Notez aussi que 40 présent 2 fois dans L est repéré à sa 1ère position trouvée dans L, à l'index 2.

10) recherche d'un élément dans une liste : opérateur ?

L ? x
Argument L à gauche : liste à explorer, à la recherche d'un élément. Argument x à droite : l'élément (non-liste) recherché.
Obtient une liste d'index désignant les éléments de la liste égaux à l'élément recherché.
LA = {"a","zer","ty","zer" }
LA ? "zer"--> {2,4}

L = {15,40,63,18,27,40,33,29,40,88}
X = {29,63,40,33,50}
alors L ? 40 --> {2,6,9} à comparer avec l'exemple du paragraphe 9).
Avec un opérande gauche de type chaîne de caractères, on obtient un nouvel opérateur intéressant travaillant sur les chaînes de caractères.
ch = "rechercher les lettres e"
ch::"e" --> {2,5,9,13,17,21,24}
Cet opérateur devient une meilleure alternative à la fonction Position, qui est désormais mentionnée "obsolète" dans la documentation mais maintenue pour compatibilité... jusqu'à ce que j'en décide autrement !

11) taille : opérateur % unaire à gauche

% donne le nombre d'éléments d'une liste
{1, 6, "zz"}% --> 3

12) conversion en chaîne de caractères : opérateur $ unaire à gauche

LA = {"a","zer","ty"}
LA$ --> "azerty"

NOTA sur les opérateurs unaires $ et %, unaires à gauche et unaires à droite.

voyez la cohérence des 4 notations unaires % et $:
$ "azerty" --> { "a","z","e","r","t","y"} et { "a","z","e","r","t","y"}$ --> "azerty"
%5 --> {1,2,3,4,5} et {1,2,3,4,5}% --> 5

13) Retire

Retire(L,IX) enlève de la liste L les éléments dont les index sont présents dans la liste IX; le retour de la fonction est la liste L modifiée.
Retire(L,ix) fait de même sur un seul élément, dont l'index est spécifié dans l'argument ix
Exemple:
Retire({"a","z","e","r","t"} , {1,3}) --> {"z","r","t"}
Pour retirer un bloc d'éléments contigus, il est plus efficace d'utiliser les décalages << et >>.
Retire({"a","z","e","r","t"} , 2) --> {"a","e","r","t"}

14) Insere

Insere(L,ix,M) ajoute à la liste L, à compter de l'index spécifié par ix, les éléments de la liste M; le retour de la fonction est la liste L modifiée. Un ajout en position 1 prends la première position 1 et décale l'ensemble des éléments vers la droite.
Exemple :
Insere({3,4,5,6},2,{"a","b"}) --> {3,"a","b",4,5,6}
Pour ajouter une liste à la fin d'une autre, il est plus efficace d'utiliser l'addition +

15) modification d'un élément, fonction Change

Change(L,i,e)
L'élément de rang i de la liste L est modifié, et lui seul, et prends la valeur e. La fonction renvoie en retour la liste L ainsi modifiée.
Change({ "a","z","e","r","t","y"},2,"ww") --> { "a","ww","e","r","t","y"}

16) modification d'une liste d'éléments

Change(L,IX,M)
IX et M sont des listes de même taille. Les éléments dont l'index est un élément de la liste IX prennent la valeur de l'élément de M de même rang. La fonction renvoie en retour la liste L ainsi modifiée.
Change({ "a","z","e","r","t","y"}, {2,6}, {"toto",1959}) --> { "a","toto","e","r","t",1959}

(2.2) OPERATIONS PORTANT SUR LES ELEMENTS DES LISTES. La notation °

Dans cette catégorie, les éléments de la liste résultat sont obtenus par l'action d'opérations portant sur les valeurs de chaque élément de la liste .

1) une notation nouvelle : la °distribution

On utilise la notation ° (rond), utilisée accolée immédiatement **devant** le nom d'un opérateur ou d'une fonction. Comme dans °Longueur() ou °+. Cette notation signale que l'action de l'opérateur ou de la fonction va s'effectuer sur chaque élément d'une ou deux listes. Par exemple, de façon intuitive, l'addition d'un élément et d'une liste : 2 °+ {1,2,3} --> {3,4,5}
En effet, une expression "élément °opérateur liste" applique l'opérateur avec comme argument 1 "élément", et pour argument 2 chaque élément de la liste.

Tout aussi intuitif , l'addition des éléments de deux listes : {"A","B"} °+ {"C","D"} = {AC","BD"}
Simplement, une expression "liste °operateur liste" applique l'opérateur sur chaque couple de même rang dans les deux listes.

Attention {1,2,3} °+ {2,4,6,8} --> {3, 6, 9}
Les trois premiers résultats sont évidents, selon les explications ci-dessus. Le quatrième exige en revanche une règle de gestion lorsqu'une des deux listes est plus courte que l'autre. Cette règle est simplement que si un des éléments est vide, le résultat de l'opération l'est aussi. Dans notre exemple, <vide> + 8 = <vide> car la liste {1,2,3} n'a pas de 4ème èlement à mettre en face de 8.

Cette notation ° s'applique à :

- n'importe quel opérateur binaire (+, -, *, &, >=, etc..) :
exemple L °+ M comme on vient de le voir,

- n'importe quel opérateur unaire (-, !, etc..), que ce soit unaire droite ou unaire gauche :
{2,3,4} °! --> {2,6,24} (distribution de la fonction "factorielle" qui est unaire à gauche)
°-{"azer", 8, 1.23 } --> { "reza",-8, -1.23 } (distribution de la fonction "opposé" qui est unaire à droite)

- les fonctions d'un seul argument, même les fonctions définies par l'utilisateur, à condition qu'elles ne renvoient qu'une ou zéro valeur en retour.
°Lettre({1,2,3}) --> {"A", "B", "C"}

- les fonctions de deux arguments, même les fonctions définies par l'utilisateur, à condition qu'elles ne renvoient qu'une ou zéro valeur en retour.
°Inclus({"AF","DEF",DG"},"DEFGH") --> {0,1,1} car AF n'est pas dans DEFGH (A manque), mais DEF et DG le sont.

2) la réduction°

On peut réduire une liste en 1 seule valeur, avec n'importe quel opérateur binaire associé à la notation °, accolée immédiatement **après** le nom de l'opérateur, comme dans +° par exemple. La réduction et son opérateur cible s'utilisent dans une position habituellement réservée aux opérateurs à un seul argument (dits opérateurs unaires). La réduction permet la généralisation d'un opérateur binaire à un nombre quelconque d'arguments (les éléments d'une liste).

par exemple
a = +°LA
signifie que l'opérateur + est exécuté itérativement sur chaque couple d'éléments de la liste, et le résultat final stocké dans a. Finalement, on a simplement sommés entre eux les éléments de la liste.
"+°" est ici utilisé en opérateur à un seul argument, en l'occurrence "LA".

LA = {3, 6, 9}:
+LA --> 3 + 6 + 9 --> 18 (somme des 3 éléments)
*° LA --> 3 * 6 * 9 --> 162 (multiplication des 3 éléments)

la réduction s'applique aussi avec les fonctions de base et même les fonctions utilisateurs, à condition qu'elles aient exactement deux valeurs d'entrée et une valeur de sortie. Par exemple

x = Fonction°(LISTE)

3) le °produit externe°

Il s'écrit "La °codop° Lb", où "codop" est un opérateur binaire ou une fonction de deux arguments. Si La est de dimension n et Lb de dimension m, alors l'opérateur produit une liste de n x m éléments, composée de tous les couples possibles "a codop b" de chacune des 2 listes. Exemple :
{2,3} °*° {1,2,3} --> {2,4,6,3,6,9}

ou avec les fonctions de deux arguments :
°Debut°({"ABC","DE","F"} , {1,2}) --> {"A","AB","D","DE","F","F"}
le produit externe s'applique aussi avec les fonctions utilisateurs, à condition qu'elles aient deux valeurs d'entrée et une ou zéro valeur de sortie. Par exemple
P = °Fonction°(L,M)

(3) ALLONS PLUS LOIN : liste de listes

Un élément de liste peut être lui même une liste. Ainsi on peut écrire
L = {{1,2,3},{4,5,6}} : L est une liste de deux éléments. Ces deux éléments sont aussi des listes, de chacun 3 éléments. L pourrait par exemple représenter un tableau de 2 lignes et 3 colonnes. L pourrait aussi, et ce serait plus adapté au concept de liste imbriquée, représenter un arbre comportant 2 branches, chacune des deux branches portant 3 feuilles. Les structures d'arbre sont très utilisées en informatique.

L'outil µ réalise des opérations qui modifient le niveau d'imbrication des éléments d'une liste :

- le regroupement : opérateur binaire µ

L µ n
Groupe les éléments d'une liste par paquets (sous-listes) d'une taille donnée. Exemples :
liste = {"AF","DEF",DG","XZ",3,4} µ 2 --> { {"AF","DEF"} , {"DG","XZ"} ,{3,4} } : regroupement par paquets de 2, ce qui crée 3 sous-listes de 2 éléments.
liste = (%12 µ3) µ2 --> { { { 1,2,3 } , { 4,5,6 } } , { { 7,8,9 } , { 10,11,12 } } } : d'abord un regroupement par paquets de 3, soit 4 sous-listes de 3 éléments, puis un nouveau regroupement par paquets de 2, soit 2 sous-listes de 2 sous-listes de 3 éléments.

- la mise à plat : opérateur unaire µ

Ramène au 1er niveau de la liste l'ensemble des éléments imbriqués dans des sous-listes, sous-sous-listes, etc.., quelque soit le niveau d'imbrication. Cet opérateur nettoie au passage la liste de toute sous-liste vide.
µ{ {"AF","DEF"} , "AZ", {"DG", {"Y","XZ"} }, { } } --> {"AF","DEF","AZ",DG","Y","XZ"}

- distribution, distribution, produit externe

Bien que parfaitement compatibles, ces opérations sont rarement utiles sur des listes de structure trop bizarres... Ce qu'il faut retenir :

soit on travaille sur une liste simple , comme déjà vu :
L = {1,2,3,4,5,6} --> +° L = 21 (addition de l'ensemble des éléments)
soit on travaille sur une liste compliquée, remise bien à plat par l'opérateur µ :
L = {1,2,3,{4,5,6}} --> +°(µL) = 21
ce qui permet d'obtenir toujours un résultat simple.

Autre exemple
L = {{1,2,3},{4,5,6}}
M = {{2,4,8},{3,6,9}}
(µL) °+ (µM) --> R = {3,6,11,7,11,15}
On peut toujours, si nécessaire ou pour faire joli, redonner au résultat la structure d'origine avec Rµ3 = {{3,6,11},{7,11,15}}.

(4) DEUX EXEMPLES DE PROGRAMMATION EN LISTE

Enoncé du problème 1
On définit la valeur d'une lettre comme étant son rang dans l'alphabet ("a" = 1, "b" = 2,..., "z" = 26). On définit la valeur d'un mot comme la somme des valeurs de ses lettres. Question : existe t-il des nombres dont la valeur de la forme littérale en lettres ("un", "deux", ... "mille") est aussi égale à la valeur de sa forme numérique (1, 2, ..., 1000) ? Pour information, de tels nombres sont appelés les nombres de Schulz.

***** programmation classique *****

On utilise naturellement les fonctions Poids, Valeur et le script déjà tout fait "Nel.eea" pour le calcul des formes littérales. Pour le reste, c'est très simple : on fait une boucle de calcul sur les nombres de 1 à 500 et on teste si la valeur obtenue est aussi la valeur de l'indice de boucle. Si oui, on affiche.

```
#include(Nel.eea)
max = 500
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2
2,23,24,25,26)

for(i=1;i<=max;i+=1)
  if (Valeur(Majus(Nel(i))) == i)
    Selection(i)
  endif
endfor
```

***** programmation en liste *****

```
#include(Nel.eea)
N = %500
Poids(1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,2
2,23,24,25,26)

Selection(N & °Valeur(°Majus(°Nel(N))) °== N)
```

Explication du programme en liste

`N = %500` est la liste des nombres de 1 à 500 : {1,2,...,500}
`°Nel(N)` est la liste des formes littérales {"un","deux",...,"cinq cents"}.
`°Majus(°Nel(N))` est la même liste en majuscules : {"UN","DEUX",...,"CINQ CENTS"}.
°Valeur(`°Majus(°Nel(N))`) est la liste des valeurs de ces formes littérales : {35,54,...,104}.
`°Valeur(°Majus(°Nel(N))) °== N)` compare cette liste, membre à membre, avec la liste des nombres de 1 à 500. Le résultat est une liste de 500 booléens (valeurs 1 ou 0) selon que les nombres sont identiques, au même rang.

`N & °Valeur(°Majus(°Nel(N))) °== N` est la sélection des nombres de la liste N correspondants aux 1 de la liste précédente, utilisée comme masque.

Enoncé du problème 2
On met 1000 petits bouts de papier, numérotés 1 à 1000, dans un grand sac. 1000 fois de suite, on tire au hasard un bout de papier du sac, on note le chiffre, on remet le bout de papier dans le sac, et on secoue bien fort. Question : écrire un programme reproduisant cette expérience, et donner le nombre de nombres différents que l'on obtient (autrement, dit, éliminer les doublons).

***** programmation classique *****

C'est basique : on fait 1000 tirages avec Hasard(n) , avec une 1ère boucle for-endfor. A chaque fois, on explore, avec une autre boucle for-enfor, les résultats précédents soigneusement conservés dans le tableau Tirage[i], pour savoir si le nombre tiré au hasard a déjà été obtenu. Si ce n'est pas le cas, on augmente nb += 1 le nombre de nombres différents.

```
n = 1000
nb = 0
for(i=1; i<=n; i += 1)
  Tirage[i] = Hasard(n)
  for (k=1; k<i; k += 1)
    if (Tirage[k] == Tirage[i])
      breakfor
    endif
  endfor
  if (k == i)
    nb += 1
  endif
endfor
Selection(nb)
```

***** programmation en liste *****

```
n = 1000
Tirage = °Hasard(n*{n})
Selection(+°(Tirage?Tirage °== %n))
```

Comparaison des temps de calcul (hors temps d'affichage)

classique : 680 millièmes de secondes
listes : 4 millièmes de secondes

le programme en liste est **170 fois plus rapide**....

Explication du programme en liste

`Tirage = °Hasard(n*{n})`
c'est très direct : 1000 * {1000} donne une liste de 1000 chiffres 1000, c'est à dire {1000,1000,...,1000} avec 1000 éléments égaux à 1000. °Hasard distribué sur cette liste, cela revient à appliquer Hasard sur chaque élément de la liste. Le résultat est une liste de 1000 nombres tirés au hasard entre 1 et 1000. Par exemple {123,872,54,927,....}, liste contenant 1000 éléments.

`Tirage?Tirage` recherche les éléments de Tirage dans la même liste Tirage. Voyons ce qui se passe avec un exemple :
Si Tirage = {123,345,234,456,321}, alors Tirage?Tirage = {1,2,3,4,5}, ce qui signifie que l'élément 1 à gauche et vu en position 1 à droite, l'élément 2 à gauche en position 2 à droite,etc... Ceci car la liste n'a aucun doublon....
En revanche, si Tirage = {123,345,123,456,321}, alors `Tirage?Tirage` = {1,2,1,4,5}. L'élément 3 à gauche (123) est vu d'abord en position 1 à droite !
Donc la différence entre la suite naturelle {1,2,3,4,5} et la suite calculée {1,2,1,4,5} nous indique l'existence d'un doublon avec l'élément de rang 3 !

`Tirage?Tirage °== %n` compare `Tirage?Tirage` avec la liste naturelle des nombres de 1 à 1000. Comme vu à l'instant, les différences (donc les 0) correspondent aux doublons. Avec notre exemple, le

résultat serait : {1,1,0,1,1} : car le 3ème élément est en doublon, repéré par un 0 en rang 3 dans le résultat. Le nombre de nombres différents est simplement le nombre de 1 dans {1,1,0,1,1}, c'est à dire 4, que l'on obtient en additionnant entre eux les éléments de la liste à l'aide d'une réduction° portant sur l'opérateur +

```
+°(Tirage?Tirage °== %n)
```

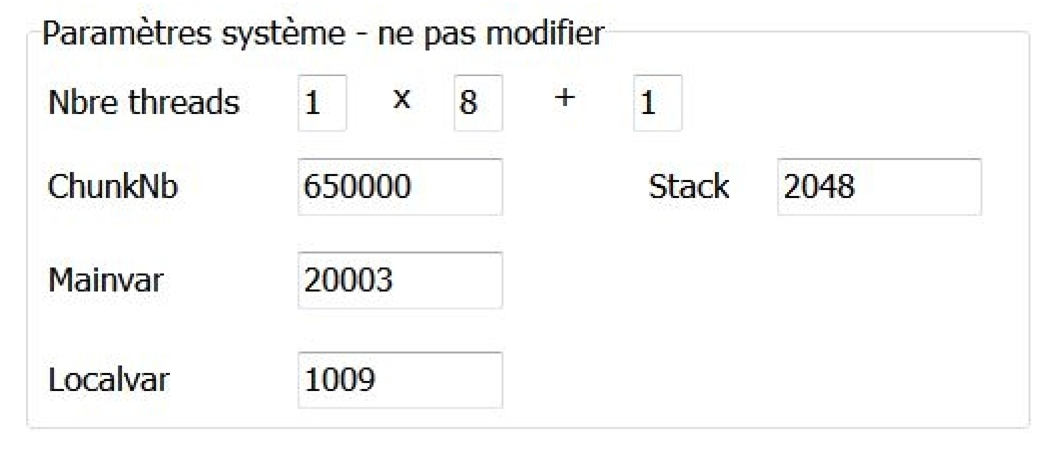
Vous voyez que la programmation en liste est une approche radicalement différente, et ne consiste pas à reproduire avec des objets listes les algorithmes classiques de la programmation structurée. En fait, la programmation en liste propose une grande variété d'algorithmes tout faits, au travers de ses opérateurs spécialisés. Le jeu consiste à les assembler astucieusement pour obtenir le résultat voulu. Une caractéristique d'un "bon" programme en liste est l'absence, si possible totale, de toute structure de bloc.

==
Ces spécifications sont très influencées par le texte de Bernard Legrand "APL vu du Ciel". (c) Interfluences 2006
==

EEA options avancées

EEA options avancées

Il est temps désormais que je dévoile les secrets des options avancées, celles qui se trouvent dans la fenêtre Options->Onglet EEA :



Nbre threads

Le nombre de threads utilisé dans les programmes EEA est limité par la valeur résultat du calcul indiqué. Dans l'exemple ci-dessus, les chiffres 1 peuvent être remplacés par des valeurs de votre choix. Le chiffre 8 quant à lui est celui correspondant au nombre de cores de votre ordinateur. Il n'est pas modifiable. La valeur par défaut, qui est donc égale à ce nombre de cores + 1, est suffisante pour presque tous les usages. Si vous voulez que votre ordinateur ne soit pas mis à genoux par EEA, réduisez le nombre (par exemple avec 0 en multiplicateur et 4 pour l'addition). Si vous programmez des fonctions thread qui s'appellent entre elles, il se peut que des situations de dead-lock se produisent à cause d'un manque de threads nécessaires pour permettre la bonne fin d'une pile d'appels. Ajustez la valeur à la profondeur de vos appels de fonctions.

ChunkNB

Sous ce nom barbare se cache un paramètre très important : celui qui correspond au nombre de "petits" emplacements mémoire pré-réservés par EEA pour l'exécution des scripts. Chaque nombre entier, chaque fragment de texte, etc.. consomme au moins un de ces emplacements. Si le nombre d'emplacements réservés est épuisé, EEA demande de la mémoire supplémentaire à Windows.... et cette opération est beaucoup moins rapide que ce que sait faire EEA, car EEA gère lui même les verrous de partage de mémoire entre ses threads. La valeur par défaut est suffisamment élevée pour la plupart des usages. Cependant, si par exemple vous travaillez sur des listes de très grande taille (fonction Anagrammeliste par exemple?), il est utile de mettre un nombre au moins aussi grand que le nombre d'éléments de vos listes. Approximativement, la mémoire réservée par EEA est de 48 octets multiplié par le nombre Chunknb. De sorte que 1.000.000 d'éléments de consomme que 48 MO de mémoire, ce qui est peu sur un ordinateur moderne.

Stack

Ce paramètre contrôle la taille d'une zone de mémoire utilisée pour le transfert de valeurs entre fonctions. Si vous utilisez des fonctions récursives qui descendent à des milliers de niveaux de profondeur (bon courage !), augmentez cette valeur. De même si vous initialisez des listes avec des centaines de valeurs, par exemple : Liste = { 1,2,3, ..., 2000}. Dans ce cas, la stack doit transporter l'intégralité des valeurs de la liste, ce qui peut justifier un ajustement à la hausse.

Mainvar, Localvar

Le nombre de variable dans EEA n'est pas limité, mais cependant une structure interne est chargée de rassembler les variables du programme principal (et aussi toutes les variables globales), et comme chaque fonction dispose de propre espace de variable, chaque fonction utilise également une structure interne qui rassemble ses variables locales. Cette structure interne est optimisée pour un nombre de variables allant, approximativement, jusqu'à 4 fois les valeurs indiquées. Mainvar pilote la taille de la structure interne dédiée au programme principal, et Localvar celle dédiée aux fonctions. Pour des raisons liées à un algorithme de hashcoding interne, les valeurs utilisées doivent impérativement être des nombres premiers, sous peine de dégradation des performances.

Mise au point de programmes EEA

La mise au point de programmes EEA

EEA est un langage de programmation suffisamment puissant pour envisager toutes sortes de programmes plus ou moins sophistiqués. De sorte que la mise au point de ces programmes peut être un exercice plus ou moins difficile. Bien que très éloignée des perfectionnements d'un véritable environnement de développement intégré (IDE), la plate forme AJL dispose quand même de fonctions d'aide à la mise au point et au débogage.

Interrompre le programme, afficher les variables, poursuivre l'exécution du programme.

Un paramétrage adéquat permet de provoquer un affichage de la valeur de toutes les variables, à chaque appui sur la touche "Interrompre", ou à chaque interruption programmée (voir ci dessous).

On peut aussi utiliser des fonctions spécifiques, voyez la rubrique d'aide associée dans le lien hypertexte proposé :

La fonction Trace

La fonction Trace :

- ne provoque pas d'interruption du programme,
- manipule (on / off) la valeur de la case à cocher "Trace", qui régit l'affichage ou non de messages informatifs à chaque affectation de variable.

La fonction Tracemessage

La fonction Tracemessage :

- ne provoque pas d'interruption du programme,
- affiche un texte toujours à la même position (champ situé entre la fenêtre du programme EEA et la zone Message), le dernier texte remplace le précédent.

La fonction Pause

La fonction Pause :

- affiche un texte en zone message,
- provoque une interruption du programme,
- la poursuite du programme est obtenue en appuyant sur le bouton "Poursuivre le script EEA".

La fonction Message

La fonction Message :

- affiche un texte en zone message,
- ne provoque pas l'interruption du programme : si la zone Message est pleine (1000 éléments), les messages les plus anciens sont effacés.

La fonction Breakpoint

La fonction Breakpoint

- permet d'interrompre l'exécution du programme dès qu'une condition de votre choix est atteinte.
- affiche un texte en zone message,
- provoque une interruption du programme,
- la poursuite du programme est obtenue en appuyant sur le bouton "Poursuivre le script EEA".

La fonction Messagebox

La fonction Messagebox :

- affiche un texte dans une fenêtre d'information qui apparaît
- suspend le programme EEA et globalement tout AJL ...
- ...jusqu'à ce que vous appuyiez sur le bouton "OK" de la fenêtre apparue. Le fenêtre et son message disparaissent et le programme se poursuit.

Que faire en cas de plantage ou de mauvais fonctionnement de AJL?

Malgré tout le soin apporté au développement de AJL et EEA, il peut arriver que des erreurs m'aient échappées lors des tests. Il se peut qu'un script EEA s'arrête anormalement avec ce message d'erreur **`Erreur interne à l'exécution de la ligne x`**, voir même, pire, qu'une erreur Windows soit générée. Il se peut qu'une fonctionnalité ne se comporte pas comme décrit dans cette documentation. Dans ces cas, prenez contact avec moi et je m'efforcerai de corriger au plus vite l'anomalie en question.